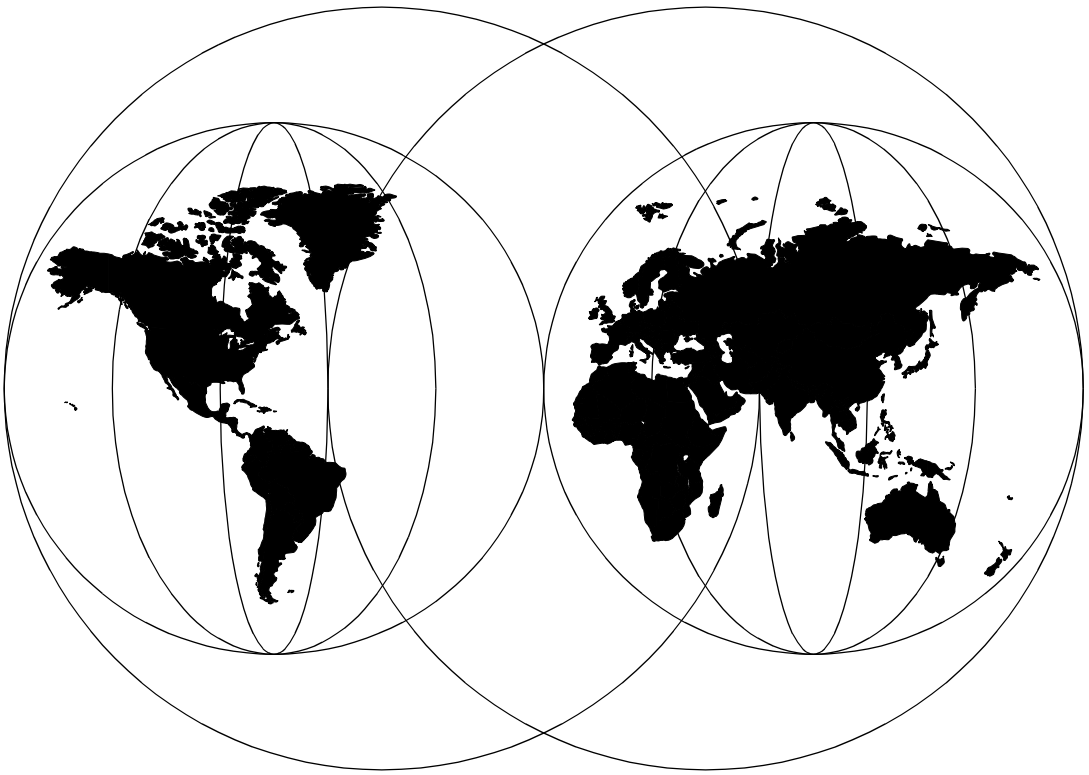
Accessing CICS Business Applications from the World Wide Web

Hanspeter Nagel *Steve Longhurst*



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-4547-02

International Technical Support Organization

SG24-4547-02

Accessing CICS Business Applications from the World Wide Web

March 1998

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix E, “Special Notices” on page 231.

Third Edition (March 1998)

This edition applies to the CICS products for use with the OS/2, Windows NT, AIX, OS390 and MVS/ESA operating systems.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

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

Contents

Figures

Tables

Preface

The advent of the World Wide Web, the availability of powerful and inexpensive personal computers, and the ready availability of connection to the Internet, led to explosive growth in Internet usage and in the number of individuals and organizations with access to it. As a result, many organizations are considering whether they should provide direct access to their operational systems for customers and/or clients via the Internet in order to provide better service or gain competitive advantage.

Notice

This document contains information such as World Wide Web document addresses that point to specific resources on the Internet. Every effort possible was made to verify the validity of this type of information as of the date that the book was sent to publishing. However, since the Internet and the World Wide Web are constantly changing, with information being added, changed, and deleted daily, it is possible that some of the information resources referenced by this document may have changed or been deleted.

This document is written primarily for information technology professionals responsible for designing and managing CICS-based OLTP applications, to help them make these applications available to users of the Internet and the World Wide Web. It looks at how this can be done, and at issues such as security and data integrity that need to be understood when deciding whether to proceed down this path, and how far to go.

How This Document Is Organized

The document is organized as follows:

- Part 1, “Introduction” on page 1 introduces some important terms and concepts from the Internet and World Wide Web, and looks at ways in which the World Wide Web differs from a traditional CICS OLTP environment. Also included is information on security in the Internet and World Wide Web environment.
- Part 2, “Programming and Connectivity” on page 31 describes what you need to do when programming to communicate with World Wide Web servers and browsers. It then discusses how you can connect the World Wide Web environment to CICS, and the ways in which communicating with the Web can affect your CICS applications.
- Part 3, “Sample Applications” on page 107 describes some sample applications that illustrate many of the issues and techniques discussed in Part 2, “Programming and Connectivity” on page 31.

The Team That Wrote This Redbook

All the editions of this redbook were produced by teams of specialists from around the world working at the International Technical Support Organization San Jose Center.

1. and 2. Edition

Guido De Simoni
International Technical Support Organization, San Jose Center

Steve Wall
IBM Hursley, U.K.

David Thrum
IBM Australia

3. Edition

Hanspeter Nagel is an Advisory Systems Engineer at the International Technical Support Organization, San Jose Center. He writes extensively and teaches IBM classes worldwide on all areas of Distributed Transaction Systems. Before joining the ITSO, Hanspeter Nagel worked in the services organization of IBM Switzerland where he was responsible for DCE, Security and Distributed Transaction Systems in several customer projects. You can reach him by e-mail at hnag@us.ibm.com

Steve Longhurst is a software developer working at IBM Hursley for the CICS/ESA Data Communications group. He is part of the CICS Web Interface development team and also implemented the MVS port of the CICS Gateway for Java. Stephen graduated in 1996 from the University of Southampton with skills in Java, TCP/IP and distributed systems. You can reach him by e-mail at slong@hursley.ibm.com.

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

Susan Malaika
IBM Santa Teresa Lab

Shirly Hentzel
International Technical Support Organization, San Jose Center

Steve Wall
IBM Hursley, U.K.

Eugene Deborin
International Technical Support Organization, San Jose Center

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 249 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

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

- Send us a note at the following address:

  `redbook@us.ibm.com`

Part 1. Introduction

Chapter 1. Introducing the Web

In this chapter we introduce some terminology and concepts of the Internet and World Wide Web (WWW, W3, or simply the Web) that may be unfamiliar if you are used to working in a traditional online transaction processing (OLTP) environment such as CICS. To fully appreciate the issues we deal with in the rest of the book, you should have some familiarity with these subjects.

We do not attempt to explain the terminology and concepts in detail. *Using the Information Super Highway* GG24-2499, provides a comprehensive introduction to the Web for those readers needing additional background information.

The Internet is a worldwide collection of interconnected computer networks that communicate using the Transmission Control Protocol/Internet Protocol (TCP/IP). The Internet for data and computers is analogous to the telephone network for voice–multiple independent networks are connected to allow access from any network to any other network.

The Internet and TCP/IP provide the infrastructure for a number of facilities and applications. Since TCP/IP was designed for communication between computers, rather than between, say, a fixed-function terminal and a host computer, many TCP/IP applications are structured using a client/server model. Because of TCP/IP's strong historical roots in UNIX, the server component of these applications is commonly referred to as a *daemon*, the UNIX name for background tasks used to service user, or client, requests.

Some of the more well-known TCP/IP applications are:

Telnet Telnet provides a virtual terminal facility that allows users of one computer to act as if they were using a terminal connected to another computer. The telnet client program communicates with the telnet daemon on the target system to provide the connection and session.

FTP The file transfer protocol FTP allows you to transfer files between computers, using a connection between an FTP client on the requesting computer and an FTP server daemon on the destination system.

Mail The Internet provides electronic mail using the Simple Mail Transfer Protocol (SMTP) and Post Office Protocol (POP). Most major electronic mail services provide a gateway to Internet mail. If you have access to internet mail, you have access to potential E-mail addresses of many millions of people.

News *Internet News*, also called *Usenet*, is a group discussion or conferencing facility. Many thousands of different news groups cover almost any subject you can imagine.

Gopher Gopher is a facility that helps you find resources on the Internet. Gopher presents you with simple character-based menus. Each menu item either represents another Gopher menu, or can take you directly to facilities or services such as viewing or downloading a file, or starting a telnet session. Because Gopher menus point to menus on other Gopher servers, you can search for resources across multiple internet systems.

WWW The Web is in some ways similar to Gopher in that it allows you to link resources spread across multiple machines connected to the Internet. The Web adds a number of significant enhancements, including an easy to use graphical user interface (GUI), full hypertext linking rather then simple menu selection, the ability to display multimedia objects (pictures, sound and movies) as well as text and a forms capability that allows two-way communication between the Web browser and the Web server. In addition, the Web browser can act as a client to a number of other Internet facilities, including FTP, Gopher, News or Java, providing a single interface that is all many users will ever need to use.

The Web provides users with a consistent and simple means of accessing a variety of media types, and has rapidly developed into a global hypermedia network. The Web is officially described as a "wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents".

The perhaps modest goal of the Web designers was to provide a tool to assist the advancement of science and education. While it may well achieve this goal, the Web is also set to revolutionize many other elements of society, including commerce and government.

The flexibility and ease of use of the Web have directly fuelled the current exponential growth of the Internet and make it necessary to have the Web as a means of accessing all applications. The following list introduces some Web terminology and components with which you need to be familiar in order to understand the issues involved.

Web browser As with many other Internet facilities, the Web uses a client/server processing model. The Web browser is the client component. Examples of Web browsers include Mosaic, Netscape Navigator, and the Microsoft Internet Explorer. Web browsers are responsible for formatting and displaying information, interacting with the user, and invoking external functions, such as telnet, or external viewers for data types that Web browsers do not directly support.
Web browsers have become the "universal client" for the GUI workstation environment, in much the same way that the ability to emulate popular terminals such as the DEC VT100 or IBM 3270 allows connectivity and access to character-based applications on a wide variety of computers. Web browsers are widely available for all popular GUI workstation platforms, and are inexpensive.

Web servers Are responsible for servicing requests for information from Web browsers. The information can be a file retrieved from the server's local disk, or it can be generated by a program called by the server to perform a specific application function.
There are a number of public-domain Web servers available for a variety of platforms including most UNIX variants, as well as personal computer environments such as OS/2 Warp and Windows NT. Some well-known public domain servers are CERN, NCSA httpd, and Apache servers (available on a variety of UNIX and non-UNIX platforms).
IBM has released the Domino Go Webserver 4.6.1. Domino Go Webserver is a scalable, high-performance Web server that is

available on OS/390 and on many workstation platforms (AIX, Solaris, HP-UX, OS/2 Warp, Windows NT, and Windows 95). It brings you state-of-the-art security, site indexing capabilities, advanced server statistics reporting, and relational database connectivity with Net.Data
Domino Go Web server is the successor to IBM's well known Internet Connection Secure Server (ICSS).

HTTP The Hypertext Transfer Protocol (HTTP) is the protocol that a Web browser uses to communicate with a Web server.

HTML The Hypertext Markup Language (HTML) defines the format and contents of documents sent from the Web server to a Web browser for formatting and display. HTML uses tags to specify formatting and document structure and identify hypertext links. It is similar to the Standard Generalized Markup Language (SGML), the ISO standard for specifying document format and structure.

URL The Uniform Resource Locator, or URL, is a standard naming convention for identifying a resource such as an HTML document, Gopher menu, or FTP file transfer. You use URLs within HTML documents to define hypertext links to other documents or resources.

Forms The initial goal of the Web designers was to provide a mechanism to allow you to find and display information on the Internet. In other words, the information flow was essentially one-way (from Web server to Web browser). Subsequently, a forms function has been added to HTML that allows you to specify input fields that enable Web browsers to be used to send data to the server. This function allows the Web browser to be used to access many different types of applications, including those that run on OLTP systems such as CICS.

CGI The Common Gateway Interface (CGI) is a means of allowing a Web server to execute a program that you provide, rather than retrieving a file. A number of popular Web servers support the CGI. For some applications, for example displaying information from a database, you must do more than simply retrieve an HTML document from a disk and send it to the Web browser. For those applications, the Web server needs to call a program to generate the HTML to be displayed. Most Web servers provide a mechanism to do this. The CGI is not the only such interface, however.

XML The Extensible Markup Language (XML) describes a class of data objects called *XML documents* which are stored on computers, and partially describes the behavior of programs that process these objects. XML is an application profile or restricted form of SGML. The goal of XML is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

CORBA The Common Object Request Broker Architecture (CORBA) allows applications to communicate with one another no matter where

they are located or who has designed them. CORBA 1.1 was introduced in 1991 by the Object Management Group (OMG) and defined the Interface Definition Language (IDL) and the Application Programming Interfaces (API) that enable client/server object interaction within a specific implementation of an Object Request Broker (ORB). CORBA 2.0, adopted in December of 1994, defines true interoperability by specifying how ORBs from different vendors can interoperate. IBM's Component Broker `http://www.software.ibm.com/ad/cb/` is a very good example of a CORBA implementation.

Java

Java is an object oriented programming language very similar to C++. The main strength of the Java programming language is that it is platform independent and easy to distribute over the Internet. Java compiled code runs on every computer, regardless of its operating system, that supports the Java virtual machine. This makes Java a very popular programming language for Web programming of today.
Java invented several new terms. These are the most important you for you to know:

Applet: A Java applet is a program written in Java that is automatically downloadable and executable by a browser or network computer.

JavaBeans: Are portable, platform-independent component models written in Java. Beans are Java classes that can be manipulated in a Visual builder tool and composed together into an application. Any Java class that adheres to certain properties and event-interface conventions can be a JavaBean.

Enterprise JavaBeans: Defines a component model for the development and deployment of Java applications based on a multitier, distributed object architecture. A component model defines an environment to support reusable application components. Components are previously developed pieces of application code that can be assembled into working application systems. Enterprise JavaBeans extends the JavaBeans component model to support server components.

Server components: Are application components that run on a server. In a multitier application architecture, most of an application's logic is moved from the client to one or more servers. A server component model simplifies the process of moving the logic to the server. The component model implements a set of automatic services to manage the component.

JVM: A Java Virtual Machine (JVM)JVM is an abstract computer instruction set. It functions like a real computer with instruction set and memory areas. The platform-specific implementation of JVM makes Java platform and operating system independent. JVM is often mentioned as the Java runtime environment and, as of today, is implemented within all major Web browsers.

RMI: Remote Method Invocation (RMI) allows you to write distributed objects using Java. RMI enables the programmer to create distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. At the most basic level, RMI is Java's remote procedure call (RPC) mechanism.

Now that you are familiar with the basics of the Web, you can look at some of the similarities and differences between the characteristics of typical Web applications and traditional OLTP applications that we discuss in Chapter 2, "Transaction Processing and the Web" on page 9.

Chapter 2. Transaction Processing and the Web

In this chapter, we look at some of the differences between a Web environment and an OLTP environment as they typically are today. You should be aware of these differences when planning to use the Web as a front end for OLTP applications.

You can easily understand why these differences exist when you consider why each technology was originally developed. The Web is designed to allow you to easily locate and *read* documents and other information that might be present on a system anywhere on the Internet. OLTP systems are designed to provide a highly reliable means to read and *update* centrally managed databases on which organizations depend for their daily operation.

There are also, however, some important similarities between Web servers and traditional transaction monitors:

- Both use standard data streams between the client and server–HTTP/HTML for Web servers, IBM 3270/DEC VT series for OLTP.
- Both schedule and process short repetitive pieces of work on behalf of many users accessing shared data concurrently.
- In both cases, the application is running on the server, while the client is dedicated to presentation.

As we move forward and start to use the Web to access traditional OLTP applications, the challenge is to combine the ease of use and flexibility of the Web with the robustness of the traditional OLTP environment.

2.1 What is a Transaction Program?

The term *transaction* is used in different ways. We give you first a more general understanding of the term *transaction* in the computing industry followed by some CICS-specific definitions.

2.1.1 General Explanation

A transaction program performs one or more functions typically on one or more shared databases. The transaction program includes a workflow control function that moves the request message from the requestor to the appropriate application program. If more than one application is needed, this workflow control function tracks the state of the request as it moves between applications. Each application typically accesses a database system that manages shared data. In turn, all these functions use the operating system underneath.

Transaction programs perform read and update requests to servers such as databases and files. These requests are grouped into transactions.

Transactions have four important properties:

- **Atomicity:** A transaction executes completely or not.
- **Consistency:** A transaction preserves the internal consistency of the database.

- **Isolation:** Although transactions may run concurrently, each transaction is executing alone.
- **Durability:** The transaction's results will not be lost in the case of a failure.

2.1.2 Two-Phase Commit

When a program updates data across more than one system, the atomicity property must be preserved, which means that either all participating systems durably do the updates or none of them do. Since all systems involved in a transaction may work independently, this is a challenging process. The two-phase commit protocol has been developed to solve this problem. Two-phase commit is coordinated by a special program called the *transaction manager*. The transaction manager is primarily a bookkeeper that writes every state of the transaction down in order to ensure atomicity of the involved recoverable resources. But how does that all work? During phase 1, the prepare phase, all recoverable resources touched by a program are updated but the physical updates are protected from other programs by locks. The transaction manager records all resource states on to a disk log. If all recoverable resources have completed their updates, the transaction manager proceeds to the second phase, notifying all resources of the success of the transaction. At this point, the recoverable resource managers release their locks and enable other programs to access the updated data. Now, the recoverable resource managers notify the transaction manager of the successful lock removal and the transaction manager returns the success back message to the application. If at any time something breaks before removing the locks, the transaction manager proceeds with a rollback procedure which brings all recoverable resources back to their original states.

2.1.3 CICS Transaction

The terminology of a CICS transaction differs slightly from the explanation above. Two-phase commit is the protocol observed when taking a synch point in a CICS Unit of Work UOW. A UOW is a sequence of processing actions, such as database changes, that must be completed before any of the individual actions performed by the transaction can be regarded as committed. After changes are committed, by successful completion of the UOW and recording of the synch point on the system log, they become durable and are not backed out in the event of a subsequent failure. A transaction program uses synch point (synchronization point) to split the actions into logically separate groups called UOWs. If failure occurs after a synch point but before the transaction program has been completed, only changes made since the most recent synch point are backed out. This leads to the fact that UOWs must be entirely logically independent, not merely with regard to protected resources, but also with regard to the execution logic.

2.2 User Expectation

While using the Web, you are typically searching for information. You might be looking for a specific piece of information, or just looking to see what is there. If you select a hypertext link, and it is not available, or the content of the link is not useful, you simply look elsewhere for something that better meets your needs.

When using an OLTP system, you are normally doing specific functions required to perform tasks that are part of your job. If a particular function does not work, you cannot successfully complete a task that is part of your job.

Clearly, the OLTP user has a much higher level of expectation about the consistency of system behavior and its reliability.

2.3 User Interface

Web browsers provide a graphical user interface that is easy to use and intuitive. You can display both large and small documents. If the document is larger than the current size of your Web browser window, you can easily scroll backward and forward through the document.

OLTP systems still often use character-based fixed function display terminals. The user interface is typically optimized for efficient use by experienced users. Screen layouts are constrained by the number of rows and columns that the terminal can display. Where more data must be displayed than can fit on a screen, you may need to issue additional transactions to retrieve and display the additional data.

2.4 Data Integrity

The Web was originally designed to allow you to retrieve and view documents available on the Web. In a read-only environment, you do not need to be concerned about recovery from failures when updating data. Consequently, Web servers do not provide facilities related to the integrity of data (HTML documents) that they access.

OLTP systems, in contrast, take responsibility for the physical and logical integrity of the data they manage, using facilities such as logging, transaction backout, and deadlock detection, either directly or in cooperation with database management systems.

OLTP systems can provide end-to-end recovery, ensuring that the user's terminal has successfully received and displayed the transaction response before database updates are committed. Web-based applications typically end before the Web server attempts to send a response to the Web browser. They have no way to tell whether the response arrived successfully.

2.5 Matters of State

In a transaction processing environment, a business function can consist of several related transactions that you perform in a particular logical sequence. The application therefore needs knowledge of the current state of processing within the logical business function that is being performed, and the ability to distinguish between different users performing the same tasks at the same time.

When you use a Web browser, each time you select a new hypertext link, you could be accessing a different Web server. The Web server itself is stateless; it treats each request separately and independently, and has no concept that a series of user interactions with the system can be related and make up a larger logical function. If you need to maintain information about the state of processing

in an application using the Web, you need either to maintain information about the state of processing with the Web browser, or maintain it on the server side.

OLTP systems, on the other hand, do understand state, and provide facilities to assist you to keep track of state within your applications. CICS, for example, allows one transaction to specify which transaction is to process the next input transaction from a user, and provides facilities such as the CICS COMMAREA and temporary storage queues to allow you to easily maintain state information and pass it from transaction to transaction.

2.6 Data Currency

When you display a document with a Web browser, you may retrieve the document directly from the server, but there are situations where you may not. Many Web browsers allow you to specify that pages and pictures are to be cached within your workstation so that objects do not need to be fetched again when you redisplay a page that you have looked at before. Similarly, if you are accessing the Web through a caching proxy gateway server, the page you display can be from the proxy gateway rather than from the server where the original document was stored, and can be an older version of that document. For additional information on firewalls and proxy gateways, see 3.2.1, “Firewalls” on page 16, and 3.2.3, “Proxy Servers” on page 18.

OLTP systems, on the other hand, work closely with database management systems to ensure that data presented to the user is current and accurate. When a user attempts to access data that is being updated, the OLTP system ensures that the data update is successfully completed and committed before sending it.

2.7 Getting Started with the Infrastructure

You can set up a Web site in a matter of hours with as little as a personal computer, modem, and connection to an Internet service provider.

It is not quite so easy to implement a traditional OLTP environment. Among other things, you probably need to:

- Select and install the hardware and software environment.
- Acquire or develop an application.
- Design and build a transaction processing network.
- Design and implement operational procedures.
- Define backup and recovery strategy and procedures.

2.8 Network Management

When you click on a hypertext link in your Web browser, you could be attempting to go anywhere in the world, to somewhere that does not yet exist, or does not exist any more. The route to the location can cause your request and its reply travel to through a number of intervening networks, all independent and separately managed. Mostly you get the response that you expect, but sometimes you do not. When you do not, there is probably not much you can do about it, because there may be no easy way to tell whether the problem is with the destination Web server, or one of the intervening networks through which your request must pass.

When you enter a transaction in an OLTP system, in contrast, your terminal is normally connected directly to that OLTP system, and all the transactions that you enter are directed to that system for processing. Many OLTP systems use dedicated private communication networks that are centrally managed along with the OLTP system itself. When a problem occurs, there is normally a help desk or similar function identified that you can contact to take responsibility for identifying and remedying any problem that has occurred.

2.9 The Importance of Being Available

Internet usage has changed dramatically during the past two years . In an earlier stage, the Internet was used almost exclusively as a huge read-only data source. Today, e-commerce (an acronym for business done through electronic media) is dominating the usage and further development of the Internet. The transition from a read-only data source to an interactive e-commerce complex has a major impact to the question of the importance of being available. Having an e-commerce Web server not available has the same impact as having a major store on a Main street unintentionaly closed.

OLTP systems such as CICS are designed to run mission-critical applications and include support for high availability or load balancing. OLTP systems are thus a perfect match for today's need of e-commerce Web applications.

2.10 Data Security

When you place information on the Web or the Internet in general, you are placing it in an environment that you must regard as insecure. OLTP systems, on the other hand, are often built on private networks where you can more easily guard the privacy of your data. As a result, today's encryption mechanisms used in Internet applications are much more advanced than those used by traditional OLTP applications.

Much work is still going on to build security facilities into the Internet and the Web so that you can safely use them for applications with stringent security requirements. See Chapter 3, "Security" on page 15 for a discussion of security issues and activities related to the Internet and the Web.

Chapter 3. Security

Of all the hot topics that arise when the Internet is discussed, security is the one that seems to provoke most discussion.

The Internet is an inherently insecure network. Unless you take specific measures to protect it, sensitive information such as credit card numbers or personal data is transmitted across the Internet, on networks over which you have no control, in a form that can easily be accessed by unauthorized users.

You have to decide whether the advantages to be gained from connecting your network to the Internet outweigh the risks. There are various tools and services that you can use to safeguard your own "trusted" network from the"untrusted" Internet.

The most secure option is, of course, not to connect your network to the Internet. However, with the World Wide Web increasingly becoming a focus for commercial activity, businesses that do not connect run the risk of conceding a competitive advantage to those businesses that do.

Our intention in this chapter is to highlight the security issues that arise when you connect to the Internet in general, and the Web in particular, and to provide guidelines to help you determine the level of security you require on your system. We also provide references and URLs pointing to documents giving more detailed information on the different aspects of security.

For a good introduction to security on the Internet, we recommend that you read at least Chapter 1 of the user's guide *IBM eNetwork Firewall*. You can find a Portable Document Format (PDF) version of this book on:

`http://www.ics.raleigh.ibm.com/firewall/support.htm`

See also the following URLs:

```
http://www.ics.raleigh.ibm.com/firewall/
http://www.alw.nih.gov/Security/security.html
http://www.semper.org/sirene/outsideworld/security.html
http://www.cert.org/
```

Before allowing Internet users access to your CICS system, you must perform a risk assessment on your own network and resources to identify the level of security you require. Some users, such as academic institutions, want their system to be as open as possible, and so have minimal security in place. Others, for example, financial institutions, want their resources to be as secure as possible.

You should use the security policy currently in place on your CICS system as the starting point for a review of your security requirements in the light of the new threat posed by the Internet. Some of the questions you should consider when deciding on your new security policy are:

- Will clients be required to log on to the Web server? If so, will they have to log on each time they make a request?
- Will clients have to use a personal identification number (PIN)?
- Will some or all of the data transmitted need to be encrypted?

- Do I want to restrict access to certain resources?
- Do I want to police only inbound traffic, or both inbound and outbound?
- Do I want the configuration of my network to be visible to the Internet?
- Do I want to log resources accessed by internal and external users?
- Do I want to restrict access to my system to specific times?

Assuming that you require some level of security for your network, you have a wide range of measures that you can take to help you implement your chosen security policy. These measures can be divided into two categories:

Access security You need to protect your computers, memory, disk, printers, and other computing equipment from unauthorized use.

Transaction security Commercial Internet applications require a means of exchanging sensitive information, such as credit card numbers, without exposing that information to unauthorized parties.

3.1 TCP/IP Layers

Before going into security in more detail, we take a brief high-level look at the TCP/IP architecture on which the Internet is built. TCP/IP is a layered architecture, as shown in Figure 1. Different approaches to security operate at different layers in the architecture. Some approaches, such as S-HTTP (see 3.3.5, “Secure Hypertext Transfer Protocol” on page 25) operate at the application layer, others such as firewalls (see 3.2.1, “Firewalls” on page 16) operate at the IP layer.

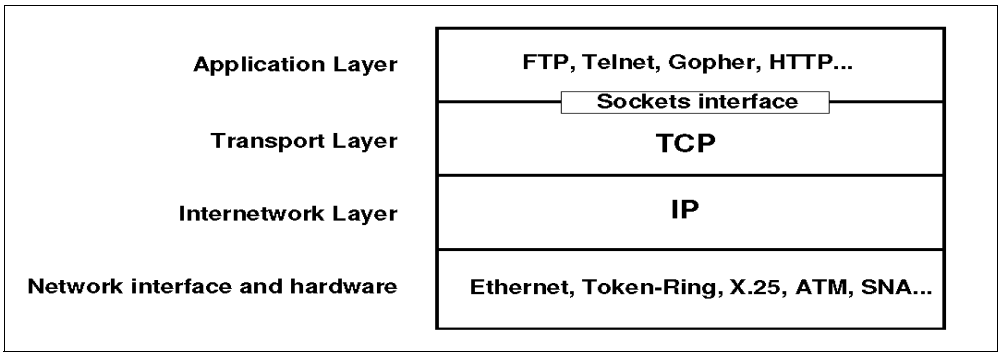


Figure 1. TCP/IP Layered Architecture

3.2 Access Security

You have at your disposal a variety of ways to limit the exposure of your hardware and software resources to unauthorized users. Below we discuss firewalls, filters, proxy servers, SOCKS (socket secure) servers, access control list files, and logging.

3.2.1 Firewalls

A firewall performs two very simple functions:

- It prevents unauthorized traffic between a trusted network and an untrusted network.
- It allows authorized traffic to flow between a trusted network and an untrusted network.

Figure 2 shows how a firewall fits between secure and unsecure networks.

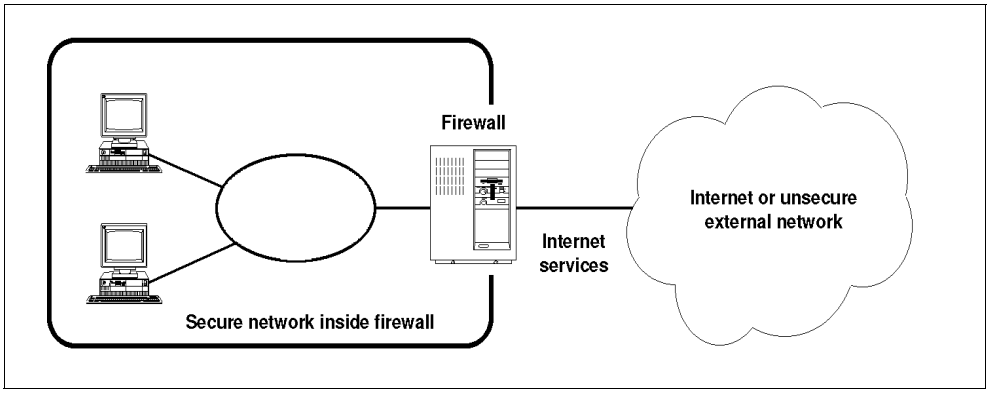


Figure 2. Firewall Protection

The machines on the trusted network shielded by the firewall are said to be inside the firewall; those on the untrusted network are outside the firewall. The firewall allows users inside the firewall to access authorized resources outside the firewall without compromising data and other resources belonging to the trusted network. The firewall prevents users outside the firewall from compromising or attacking the trusted network.

A firewall is not one specific piece of software or hardware. It is a bundle of software utilities, and a suggested network configuration, which combine to provide protection to your trusted network.

The firewall uses the following methods to control access between the trusted network and the Internet:

- Filters
- Proxy servers
- SOCKS servers.

The way in which these services are used varies greatly from one implementation to the next. However, here some features of a good firewall design:

- Anything not explicitly permitted should default to "denied".
- Keep things simple. The more complex the implementation, the greater the likelihood of bugs that present opportunities to hackers.
- Run your firewall on a dedicated machine. Do not run other applications on that machine.
- Your firewall machine should be physically secure.
- Log everything so that you have a record of what has been accessed and by whom.

We suggest that you refer to the following related publications for a more detailed description of the function of firewalls.

- Trusted Information Systems Inc.

  `http://www.tis.com/Home/NetworkSecurity/Firewalls/Firewalls.html`

- *Building a Firewall With the NetSP Secured Network Gateway*
- *Using the Information Super Highway*

3.2.2 Filters

Filters analyze data passed to the firewall on which they are running and decide whether or not that data should be allowed to continue on to its destination. Filters act either on the packets flowing across the network or at the transport layer, where connections to particular Internet Protocol (IP) addresses, or ports, can be prevented. Filters can operate on both data flowing into the trusted network and data flowing out of the trusted network.

For more details about the various kinds of filters see *Using the Information Super Highway*. For a sample implementation of filters see *Building a Firewall With the NetSP Secured Network Gateway*.

3.2.3 Proxy Servers

Webster's dictionary defines a proxy as "an authorized agent for another". Proxy servers are exactly that. They are secure servers running inside a firewall. They are authorized to make requests on behalf of another machine in the network that is itself not permitted to access resources outside the trusted network. In effect the proxy server is a gateway through which data passing between the trusted network and the untrusted network must flow, as shown in Figure 3.

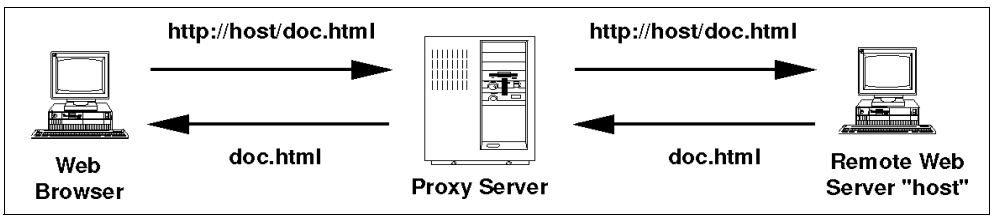


Figure 3. Using a Proxy Server

The proxy to be used is defined when the Web browser is configured. Other than that, the existence of the proxy is transparent to the Web browser inside the firewall. As far as the Web server outside the firewall is concerned, the proxy server is the Web browser. The use of a proxy therefore hides the configuration of the network inside the firewall from prying eyes.

Proxy servers operate at the TCP/IP application layer, and are application specific. For example, a proxy Web server provides proxy services on behalf of Web browsers, but does not provide proxy services for telnet clients.

Proxy Web servers can improve the performance of the network by caching frequently requested data. They do not have to send a request across the network each time the data is requested by a local Web browser; they can simply retrieve it from local storage.

For a more detailed discussion of the implementation of proxies see *Using the Information Super Highway*, and *Managing Internet Information Services*.

3.2.4 SOCKS Servers

SOCKS (Socket Secure) servers perform a similar function to proxy servers but in a different way.

SOCKS is a protocol that relays TCP sessions at a firewall host to allow applications transparent access through the firewall. SOCKS servers operate at a lower layer than proxy servers, at the transport layer, and can be used for many different services, such as telnet, FTP, gopher, and the Web. Access control using user ID and password can be applied at the beginning of each TCP session. After that, the server simply relays the data between the client and the application server, incurring minimum processing overhead. Figure 4 shows how you can use a SOCKS server to provide controlled access for your users to the Internet.

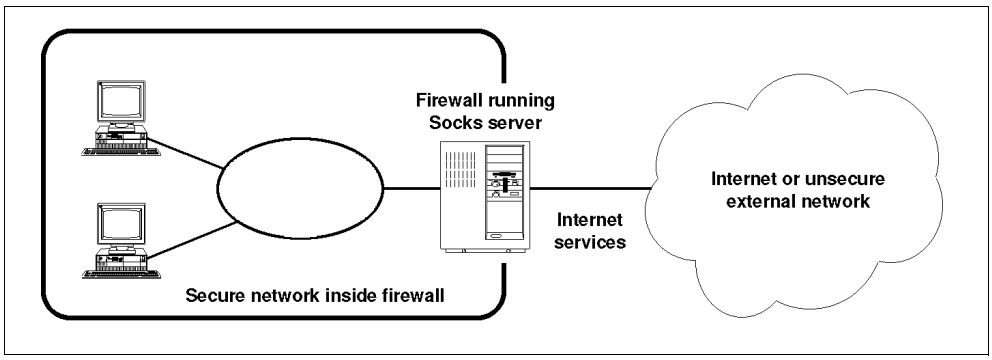


Figure 4. Using a SOCKS Server

Client programs must first be made socket secure by undergoing some minor changes and being compiled and linked using the SOCKS library. SOCKS servers are smaller and more efficient than proxy servers. Many new Internet applications now come supplied with socket-secure versions built into the product.

Because SOCKS servers are application-independent, they do not provide application-specific functions (for example, proxy Web server caching) that proxy servers can provide.

SOCKS code and documentation are available from the following FTP site:

`ftp://ftp.nec.com/pub/security/socks.cstc`

3.2.5 Access Control List Files

Access control list (ACL) files contain information that can be used by the Web server to identify which users have access to which files. Each directory in a file structure can have its own ACL.

ACLs are the Conseil Europeen pour la Recherche Nucleaire (CERN) Web server implementation of access controls. The National Center of Supercomputer Applications (NCSA) Web server equivalent is the access control file (ACF).

For more details about implementing ACLs see *Using the Information Super Highway*.

For an explanation of the two methods of server access protection, see *Spinning the Web*.

3.2.6 Logging

Logging provides a level of security by the simple expedient of recording any activity involving the integrity of your trusted network. When a user from outside your trusted network asks for access to resources inside the trusted network or a trusted user accesses resources outside the trusted network, you should record:

Origin of the request

Time of the request

Destination of the request

Whether it was authorized or not

If authorized, the resources accessed during execution of the request

When the request ended.

With accurate logging, the network administrator can trace how a security exposure occurred, and may be able to prevent it from happening again.

Logging should not be a new concept for those familiar with CICS and other OLTP systems.

3.3 Transaction Security

There are two aspects to transaction security:

- **Authentication**. For some users of the Web (in electronic commerce, for example) it is important that users authenticate themselves to Web servers, that Web servers authenticate themselves to users, or that both authenticate to each other. Whatever the form of authentication, it must not be easily compromised.
- **Encryption**. For applications in which Web clients and servers exchange sensitive information, such as user ID and password pairs, credit card details, or employee records, eavesdropping must be prevented through appropriate cryptographic techniques.

We look briefly at two of the security packages that have been around for a while (see 3.3.6, “Pretty Good Privacy” on page 25 and 3.3.7, “Kerberos” on page 25), and at two transaction security standards that are likely to become integral parts of a future integrated security solution for the Web: secure HTTP (see 3.3.5, “Secure Hypertext Transfer Protocol” on page 25), and the secure sockets layer (SSL) (see 3.3.4, “Secure Sockets Layer” on page 24).

3.3.1 Authentication

User IDs and passwords are the most obvious way for a Web server to authenticate data received from a Web browser. You can set up your Web server to control access to resources at the directory level on the server, and require users to provide a valid user ID and password before accessing these resources.

The HTTP protocol allows Web browsers and servers to exchange password and user ID information by the use of special HTTP headers. For a discussion of the current HTTP authorization architecture, see the following URL:

```
http://www.w3.org/hypertext/WWW/Protocols/HTTP/HTRQ_Headers.html
```

Figure 5 shows the basic Web authorization flows.

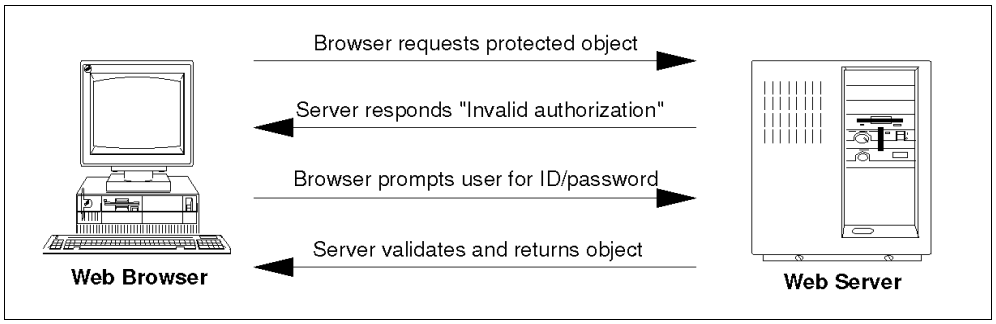


Figure 5. Password Protection of Web Documents

The sequence is as follows:

1. Web browser sends request for document to Web server for the first time with no authorization header.
2. Web server sends back the response that the authorization code is invalid.
3. Web browser retrieves user ID and password information (if known), or prompts user to supply it, and sends new request including user ID and password pair in HTTP authorization header.
4. If user ID and password are valid, Web server returns the requested document, or else returns the response that the authorization code is invalid.

We do not recommend the basic level of authorization for anyone who is concerned about security. It merely encodes the user ID and password pair rather than using encryption, making it only slightly more secure than sending user ID and password in clear text.

The proposed standard allows for public key encryption of the user ID and password (and optionally all data). Two of the security schemes that might use the proposed public key standard are Kerberos and Pretty Good Privacy (PGP).

3.3.2 Encryption Techniques

User IDs and confidential passwords are fine, but the passwords do not remain confidential for long if they are sent across the network in clear text for anyone to see. Cryptographic protocolls allows the transformation, or scrambling, of all or only part of the data to be send into an unreadable format using a mathematical algorithm. These encrypted messages are decrypted using a secret key by the recipient of the message. There are three encryption schemes:

- Symmetric Key Encryption
- Public Key
- Secure Hash Function

Symmetric Key

In the symmetric key scheme, both parties share the same secret key, which they do not divulge to others. They are the only parties able to encrypt and decrypt communications between them. Data which are encrypted with a secret key can only be decrypted with the same secret key.

Because encription and decription is always done with the same key, this is the best performing encryption technique. However, the fact that the keys must be kept secret, raises one of the main implementation difficulties with symmetric key systems. Since both keys must be kept strictly secret, the key distribution In a many-to-many environment (establishing a symmetric relationship between each pair of communicating partners) requires the exchange of a number of keys which quickly becomes unmanageable. A symmetric key scheme also needs to change keys frequently; otherwise, a hacker has unlimited time to crack the secret key. As a result, a special infrastructure is needed, as for example provided with DCE/Kerberos, to properly apply symmetric key techniques.

The Data Encryption Standard (DES) is the standard Federal secret-key algorithm. It is the most widely used commercial cryptographic algorithm in the world based on a symmetrical key scheme. DES has a long history, in the course of which no weaknesses have been discovered that would permit anything other than exhaustive attacks to discover keys by brute force. Thus, an attacker must devote a greath deal of time or resources to decipher data without the key.

The strength of DES relies on two factors. First, the algorithm itself is public and has been widely tested. Second, the strength of the algorithm increases according to the length of the keys used. The increase in strength is exponential, not linear.

Public Key

In the public key scheme, encryption involves two different keys; the public key and the private key. The key owner maintains his or her private key secretly and gives the public key to other users, enabling them to exchange encrypted messages with the key owner. Data encrypted with a public key can be decrypted only with the corresponding private key. A message that has been encrypted with a public key can be decrypted with its corresponding private key, but not with the same public key used to encrypt it.

What gives public key systems such potential is that the public key can be safely published without compromising the security of the data encrypted under it. This is because there is a complex mathematical relation between the public and private keys. For example,

1. Anne and Frank have both published their public keys to each other.
2. Frank wants to send Anne a secret message. To do so he takes her public key and encrypts the message before sending it to Anne.
3. Anne receives the message from Frank. She detects that the message is encrypted. Therfor she takes her private key and decrypts Frank’s message.

When this is so easy, why doesn’t everybody use this technique? As of today, all public key systems have one major disadvantage over symmetric key systems like DES. Encrypting small sets of data is okay, but when large datasets have to be crypted, this technique is to slow.

Like firewalls, encryption and cryptography are the subjects of much discussion and academic research. We do not intend to say much more, other than to point you to some useful URLs:

- Some useful pointers are given in:

  `http://www.yahoo.com/Computers/Security_and_Encryption`
  `http://www.semper.org/sirene/outsideworld/security.html`

- An introduction to the U.S. legal issues is presented in:

  `http://info.acm.org/REPORTS/ACM_CRYPTO_STUDY/_WEB/contents.html`

The quality of the encryption which the industry can provide is limited by U.S. export requirements that restrict the length of the encryption keys be used in commercial products which might be exported outside the U.S. As long as this situation remains the same, it is impossible to make any encryption totally secure. The best encryption that even the newest commercial security packages can provide is not unbreakable if the hacker has enough time and computer power. However, commercial encryption can raise the cost in time and computing power required to break a code until it is worth much more than the potential financial gain to the code breaker.

3.3.3 Other important Security Terms

There are many terms used around cryptography. In the following we concentrate on the terms that seem most important today. For more detailed reading see:

`http://www.semper.org/sirene/outsideworld/security.html`

Secure Hash Function

A hash function is a computation that takes a variable-size input and returns a fixed-size string, called the *hash value*. A secure hash function is a one-way process, also called a *message-digest* function, whose result is a message digest. It is impossible, or at least extremly difficult to reconstruct the original data from the hashed result. The hashed result is not predictable. With a message digest, a receiver can always check whether a message has been altered or not. You can think of a message digest as a "digital fingerprint" of the larger document. Implementations of well-known secure hash functions are MD4 and MD5.

Digital Signature

A digital signature is a mechanism that associates data with the owner of a particular private key. The digital signature for a message is created by hashing the message to produce a message digest. This message digest is then encrypted with the private key of the individual sending the message. The result then becomes the digital signature. When the message is received, the recipient decrypts the digital signature with the public key of the sender. The message digest is then calculated from the received message and compared with the decrypted one. If the two match, then the message has not been tampered with. By using the public key of the sender to verify the message, we can be sure that the message was encoded by the private key known only to the sender.

Digital Certificates

These certificates are digital documents attesting to the binding of a public key to an individual or other entity. They allow verification of the claim that a given public

key does in fact belong to a given individual. Certificates help prevent someone from using a faked key to impersonate someone else. In their simplest form, certificates contain a public key and a name. As commonly used, they also contain the expiration date of the key, the name of the certifying authority that issued the certificate, the serial number of the certificate, and perhaps other information. Most important, the digital certificate contains the digital signature of the certificate issuer.

Certification Authority
Certificates are issued by a Certification Authority (CA), which can be any trusted central administration willing to vouch for the identities of those to whom it issues certificates. In order to prevent forged certificates, the CA's public key must be trustworthy. A CA must either publicize its public key or provide a certificate from a higher-level CA attesting to the validity of its public key. It is essential that the pivate key of a CA be kept absolutely secret. Otherwise, all issued certificates are compromised.

Digital Envelope
A digital envelope is a special protocol that combines the strength of the public key method with the best of the symmetric key method. RSA invented this protocol known as the RSA Digital Envelop. Following is an example how the RSA Digital Envelop works:

Suppose Alice wishes to send an encrypted message to Bob. She first encrypts the message with DES, using a randomly chosen DES key. Then she looks up Bob's public key and uses it to encrypt the DES key. The DES-encrypted message and the RSA-encrypted DES key together form the RSA digital envelop and are sent to Bob. Upon receiving the digital envelop, Bob decrypts the DES key with his private key, then uses the DES key to decrypt the message itself.

This combines the easy key management of the public key methode with the high performance of DES encryption.

3.3.4 Secure Sockets Layer

SSL is a transport-layer security mechanism developed by Netscape Communications Corporation. It makes the TCP/IP connection between the Web browser and the Web server secure, that is, any data flowing over that connection is safe. Like the SOCKS protocol used in firewalls, SSL sits below the application layer at the transport layer, so it can be used by a variety of TCP/IP applications such as FTP and telnet, not just for the Web.

SSL has architected flows that allow:

- Clients to authenticate the server
- Servers to authenticate the client
- Negotiation of an encryption algorithm and session key
- Encryption of some or all data passed between client and server.

For the full specification of the SSL protocol, see the following URL:

`http://www.netscape.com/newsref/std/SSL.html`

The IBM Internet Connection Secure Server for AIX and OS/2 Warp are Web servers that support SSL. The IBM Internet Connection Secure WebExplorer for OS/2 Warp is a Web browser that supports SSL.

3.3.5 Secure Hypertext Transfer Protocol

Secure HTTP (S-HTTP) is a secure message-oriented communication protocol designed for use in conjunction with HTTP. It is designed to coexist with HTTP's messaging model and to be easily integrated with HTTP applications. Several cryptographic message format standards may be incorporated into S-HTTP clients and servers

S-HTTP is being proposed to the Internet Engineering Task Force (IETF) as a standard for the Internet. It was written by Enterprise Integration Technologies (EIT). Unlike SSL, which can be used for a variety of Internet applications, beside the Web, S-HTTP operates at the TCP/IP application layer and is Web-specific. It provides the following services:

Signature

Authentication

Encryption

Any data flowing between the Web browser and the Web server can be sent using any of the above services.

Like SSL, S-HTTP allows authentication to be done before any data has flowed, avoiding the overhead of an initial request from a Web browser being rejected with an authorization error.

For a description of S-HTTP, see the following URL:

`http://www.terisa.com/shttp`

The full IETF draft specification is available at:

`http://www.terisa.com/shttp/current.txt`

3.3.6 Pretty Good Privacy

Pretty Good Privacy is an encryption tool that has attracted much publicity and is available in the public domain. It is used mostly by private individuals rather than in the commercial environment. It is however being looked at by NCSA as a possible HTTP public key encryption scheme. For more details, see the following URL:

`http://hoohoo.ncsa.uiuc.edu/docs/PEMPGP.html`

3.3.7 Kerberos

Kerberos was originally developed by the Massachusetts Institute of Technology (MIT), and has for some time been widely accepted in the UNIX world as an industrial-strength security solution. It is the security solution used by the Open Software Foundation (OSF) in its blueprint for distributed computing, the Distributed Computing Environment (DCE). As such, it is available on a variety of platforms, including OS/2 Warp, MVS/ESA, OS390, Windows NT, Windows 95,

AS400, and UNIX Systems and is now making its presence felt in the world of OLTP.

Quite a lot of activity is being devoted to adapting DCE to the Web environment, which may benefit those sites that already have a DCE infrastructure and attract others who want to implement a companywide cross-platform and cross-system security infrastructure.

For more information on Kerberos and DCE, see the following URLs:

```
http://www.yahoo.com/Computers_and_Internet/Security_and_Encryption/Kerberos/
http://www.osf.org/comm/lit/lit-dce.html
http://www.osf.org/www/dceweb/DCE-Web-Home-Page.html
http://snapple.ncsa.uiuc.edu/adam/khttp/intro.html
```

3.4 Commercial Activities on the Web

Given the phenomenal growth of commercial activity on the Web, the need for a complete solution to the security issues we discuss here has been recognized for some time. While SSL and S-HTTP provide the infrastructure for secure commercial transactions over the Web, the race is on to provide an integrated security solution that allows customers, merchants, and financial institutions to conduct their business in a secure environment.

Below we look at some of the leading technologies that allow secure Internet commerce. The trend seems to be one of convergence, with the different offerings tending toward a common approach and offering complementary products.

3.4.1 SecureWeb

SecureWeb was developed by Terisa Systems, a company specializing in technologies to make *secure Internet transactions*. Terisa Systems is owned by a consortium of online service providers and Internet technology developers, including IBM, Netscape, and EIT (the developers of S-HTTP).

SecureWeb is essentially a toolkit that allows developers of Web browsers and Web servers to write products that can conduct secure transactions over the Web.

iKP (see 3.4.2, “Internet Keyed Payment Protocol” on page 26) uses SecureWeb to conduct its secure transactions. For more information see the following URLs:

```
http://www.terisa.com/pr/120296.html
http://www.ibm.com/Features/InetWorld95/pr4.html
```

3.4.2 Internet Keyed Payment Protocol

Internet Keyed Payment Protocol iKP is a multiparty security protocol developed by IBM. It allows buyers and sellers to engage third parties, such as banks and credit card companies, in a single, secure payment transaction. Buyers can securely send an encrypted credit card number to a seller, who then forwards it to the credit card company for decryption and transaction approval. The credit card company then notifies the seller of credit approval without the seller ever seeing the unscrambled buyer's credit card number.

iKP is an open proposal and has been designed with the intention of serving as a starting point for eventual standards on secure electronic payment. It received favorable responses from both the financial and technical communities and is being proposed as an open Internet secure payments standard.

For more information, see URL:

```
http://www.zurich.ibm.com/Technology/Security/extern/ecommerce/iKP.html
```

3.4.3 Secure Internet Payment Service

Like iKP, the Secure Internet Payment Service offers an integrated approach to electronic commerce. It has been developed by CyberCash Incorporated, who also have links with EIT, the developers of S-HTTP. For details, see the following URL:

```
http://www.cybercash.com/
```

3.4.4 Secure Electronic Payment Protocol

The Secure Electronic Payment Protocol (SEPP) is an open specification for secure bank card transactions over the Internet developed by IBM, Netscape, GTE, CyberCash and MasterCard. A draft version was released for comment in November 1995. SEPP provides an embodiment of the iKP protocol.

3.4.5 Secure Transaction Technology

Secure Transaction Technology (STT) is a secure payment protocol developed by Microsoft and Visa International. STT is intended for ordering products over networks, including the Internet, where payment is by bank card. STT includes messages for ordering goods and services electronically, requesting authorization of payment, and requesting "credentials" (that is, certificates) binding public keys to identities, among other services. All parties have a public/private key pair; authentication of all parties, based on their credentials and a digital signature, is a requirement of the protocol. This includes merchants, payment servers, and cardholders.

3.4.6 Secure Electronic Transaction

Secure Electronic Transaction (SET) is a specification designed to utilize technology for authenticating the parties involved in electronic credit card payments on any type of online network, including the Internet. SET was developed by Visa and MasterCard, with participation from leading technology companies, including IBM, Netscape, Microsoft, SAIC, GTE, RSA, Terisa Systems and VeriSign. By using sophisticated cryptographic techniques, SET will make cyberspace a safer place for conducting business and is expected to boost consumer confidence in electronic commerce. SET focuses on maintaining confidentiality of information, ensuring message integrity, and authenticating the parties involved in a transaction. The significance of SET, over existing Internet security protocols, is found in the use of digital certificates. Digital certificates will be used to authenticate all the parties involved in a transaction. SET will provide those in the virtual world with the same level of trust and confidence a consumer has today when making a purchase anywhere in the physical world. The SET specification is open and free to anyone who wishes to use it to develop SET-compliant software for buying or selling in cyberspace. SET evolved from

iKP, SEPP, and STT and seems to become the favorite Internet credit payment standard.

For more information on SET, check:

`http://www.visa.com/cgi-bin/vee/nt/ecomm/set/`

3.5 Java Security

The term *Java Security* is sometimes confusing. Often, people associate it with the Java security functions provided as features of the Java language and the Java virtual machine. Others may understand Java Security as their own security policies implemented on top of these features. Chapter 4.6, “Java Security” on page 49 describes the Java security functions and its implementations. In this section, we give you a more general understanding about the Java security features.

3.5.1 Security Implications while Distributing Executable Code

Java applets are small pieces of executable code that may be included in Web pages, downloaded from the net, and executed in the users Web browser. Applets exploit client/server computing dramatically; see Chapter 4.5, “Java” on page 47.

While applets solve many of the important problems in client/server and network computing (as for example distributed software management or platform independence), they also raise new concerns about security. In traditional environments, companies could protect themselves by controlling physical and network access to their computers by establishing policies for the kinds of software that can be used on their machines. These steps include building a firewall between the Internet and the company’s intranet, obtaining software only from known and trusted sources, and using antivirus programs to check all new software. Use of applets potentially adds a new security vulnerability. An employee searching an external Web site for information might inadvertently load and execute an applet without being aware that the site contains executable code. This automatic distribution of executables makes it very likely that software will be obtained from untrusted third parties. Since the applet is imported into the user’s Web browser and runs locally, this software could potentially steal or damage information stored in the user’s machine on a network file server. Also, since this software is already behind the company’s firewall, the applet could attack other unprotected machines on a corporate intranet. Such attacks would not be stopped by traditional security measures.

Web browsers protect their users from these dangers by placing strict limits on applets. Applets cannot read from or write to the local disk. Stand-alone windows created by applets are clearly labeled as being owned by untrusted software. These limits prevent malicious applets from stealing information, spreading viruses, or acting as Trojan horses. Applets are also prohibited from making network connections to other computers on the corporate intranet. This prevents malicious applets from exploiting security flaws that might exist behind the firewall or in the underlying operating system. While Java is not the first or only platform that claims to be secure in means of distributing executable code over the internet, it is perhaps the best known and most widely used.

3.5.2 The Java Security Features

In Chapter 4.6, “Java Security” on page 49, we describe what Java security features are available and how they can be applied. In the following, we give you a more general overview of them and explain the commonly used terms.

3.5.2.1 The Sandbox

Java’s security allows a user to import and run applets from the Web or an intranet without undue risk to the user’s machine. The applet’s actions are restricted to its “sandbox”, an area of the Web browser dedicated to that applet. The applet may do anything it wants within its sandbox, but cannot read or alter any data outside of its sandbox. The sandbox model is to run untrusted code in a trusted environment so that if a user accidentally imports a hostile applet, that applet cannot damage the local machine. The sandbox is made up of several different components:

The Class Loader

The class loader is the first link in the security chain. It defines which other resources from a virtual machine can be accessed by an applet. The class loader enforces the Java name space hierarchy and guarantees that a unique name space exists for classes that come from the local file system, and that a unique name space exists for each network source. When a browser loads an applet over the net, that applet's classes are placed in a private name space associated with the applet's origin. Thus, applets loaded from different network sources are partitioned from one another.

The Verifier

The verifier is invoked by the class loader. It checks to see that the applet conforms to the Java language specification and that there are no violations of the Java language rules or name space restrictions.

The Security Manager

The security manager enforces the boundaries around the sandbox. Whenever an applet tries to perform an action that could corrupt the local machine or access information, the Java Virtual Machine first asks the security manager if this action can be performed safely. If the security manager approves the action (for example, a trusted applet from the local disk may be trying to read the disk, or an imported untrusted applet may be trying to connect back to its home server) the virtual machine will then perform the action. Otherwise, the virtual machine raises a security exception and writes an error to the Java console. The security manager will not allow an untrusted applet to read or write to a file, delete a file, get any information about a file, execute operating system commands or native code, load a library, or establish a network connection to any machine other than the applet’s home server.

An application or a web browser can only have one security manager. This assures that all access checks are made by a single security manager enforcing a single security policy. The security manager is loaded at start-up and cannot be extended, overridden, or replaced. For obvious reasons, applets cannot create their own security managers.

3.5.2.2 Extended Java Security Facilities

The sandbox model described above protects the end-user’s machine and networked computing resources from damage or theft by a malicious applet.

Users can run untrusted code obtained from the network without undue risk to their system. The sandbox model does not address such security and privacy issues as:

- Authentication
- Digital Signature
- Auditing
- Encryption

These security and privacy issues are subject of SUN's ongoing Java standardization and development process. The first release of Java Security in JDK 1.1 contains a subset of cryptography functionality, including APIs for digital signatures and message digests. In addition, there are abstract interfaces for key management and certificate management. A good place to find the latest information about Java security is:

`http://java.sun.com/security/`

Part 2. Programming and Connectivity

Chapter 4. Programming for the Web

Many methods have evolved to access Business Applications from a Web browser. The extent to which you need to program for the Web depends on what you intend to use the Web for. If you are using the Web to store and retrieve preformatted HTML pages, then you don't need to do any Web programming at all. Everything is handled by your Web server and Web browser.

Using the Web to access existing business applications without modifying the business applications, may require additional programming. The additional programming can either be done behind an interface that allows the Web server to execute a program and return the results of that program (if required) to the Web browser, or through a Java client/server scheme.

Most Web servers provide an interface that allows the Web server to execute a program and return the results of that program (if required) to the Web browser. The most commonly used interface is the Common Gateway Interface (CGI), described in 4.3, "Common Gateway Interface Scripts" on page 39. The CGI is an architected way of invoking programs from a Web server and returning any output from that program to the Web browser. The CERN, the NCSA, and the Domino Go Webserver use the CGI interface. The GoServe Web server has an interface that provides a similar function, but it uses filter programs rather than a CGI-style interface.

In the Java client/server approach, the Web server downloads Java classes required to access business applications and/or business data from a server system directly. In such a case, the Web server delivers the access programs for any business applications or business data through the Web to the requesting Web clients which are Java client enabled. With this techniqe the designer of a Web business application access program is free to use any client /server model and the Web client runs always the latest code version.

Allowing Web servers to invoke or distribute programs transforms the Web from a data retrieval and display tool with limited scope into an extremely powerful and flexible method of client/server transaction processing that can take user input entered at the Web browser (the client), and pass that user input to programs running on the Web server. By using Web server programs in this way, you can perform such tasks as:

- Create HTML pages dynamically in response to user input.
- Perform administration functions on your Web server machine remotely.
- Create, update, and delete files on your Web server machine
- Execute traditional style short-running transactions.
- Issue SQL queries to a relational database manager (RDBM) such as DB2 or Oracle.
- Access any application that manages shared data, for example Lotus Notes and IBM BookManager BookServer.
- Create a smarter client while using downloaded Java classes.

4.1 Using Uniform Resource Locators

It is worth taking some time to look at URLs in more detail, because you need to understand how they work so you can write programs for the Web. URLs

represent the hypermedia links and links to network services within HTML documents. You can represent nearly any file or service on the Internet with a URL.

A URL can be divided into three distinct sections, as shown in the example in Figure 6.

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

Figure 6. A Uniform Resource Locator

The first part of the URL `http://` specifies the method, service, or protocol to use to access a Web document. This is `http`, for HTML documents but the Web also supports other Internet protocols including `ftp`, `telnet`, `news` and `gopher`. You can also specify an access method of `file` to allow your Web browser to access a file directly from a disk on your personal computer.

The second part `www.ibm.com` is the Internet address of the computer on which the the data or service is located. If your domain name server cannot resolve the name, you can also specify the TCP/IP address directly, for example `9.20.2.35`. You may also need to append the TCP/IP port to which the request is to be sent, for example `www.ibm.com:80` or `9.20.2.35:80`. You need to do this only if the server is not using the standard or well-known TCP/IP port assigned to that service. The standard port for http is 80.

The third part `Security/content.html` specifies the name of the file or service that is being requested. In CICS terms, this part of the URL can be loosely described as the Web equivalent of the CICS transaction identifier.

Sites that run World Wide Web servers often include `www` as the first part of their Internet address.

Here are some examples of URLs:

- `http://www.research.ibm.com/music/music3.html`
  Lets you retrieve a sound file and plays it
- `http://www.internet.ibm.com/computers/networkstation/download.html`
  Lets you retrieve brochures and display them, either in a separate program (PDF reader) or within a hypermedia document
- `file:///c:/www/html`
  Displays the contents of the `C:\www\html` directory on your own workstation
- `http://www.redbooks.ibm.com/catalog/trnsactn.htm`
  Connects to an HTTP server and retrieves an HTML file
- `ftp://ftp.transarc.com/welcome.msg`
  Opens an FTP connection to `www.xerox.com` and retrieves a text file
- `gopher://www.hcc.hawaii.edu`
  Connects to the gopher at `www.hcc.hawaii.edu`
- `telnet://www.hcc.hawaii.edu:1234`
  Open a telnet session to `www.hcc.hawaii.edu` at port 1234
- `news:alt.hypertext`
  Reads the latest Usenet news by connecting to a user-specified news host and returns the articles in the `alt.hypertext` newsgroup in hypermedia format.

Most Web browsers allow you to specify a URL and connect to that document or service. When selecting a hypertext link in an HTML document, you are actually sending a request to open a URL. In this way, hyperlinks can be made not only to other texts and media, but also to other network services. Web browsers are not simply Web clients, but can also be FTP, gopher, and News clients.

Be aware that, depending on the file system being used by the Web server, URLs are most commonly case-sensitive!

4.2 Hypertext Transfer Protocol Header Information

Any request received from a Web browser has HTTP header information that can be very useful both for logging and auditing purposes, and for making your Web programming easier and more user-friendly. The header consists of four sections, the general header, the request header, the response header and the entitity header. Table 1 to Table 4 shows the current HTTP1.1 architected headers.

You can use the *cgiutils* tool (see 4.9, “Utility CGIUTILS” on page 75) to help you to build HTTP headers. For full details of the HTTP1.1 specifications, see the following documents:

`http://www.w3.org/Protocols/Specs.html#HTTP1.1`
`http://www.w3.org/Protocols/rfc2068/rfc2068`
`http://www.w3.org/Protocols/HTTP/Issues/`

4.2.1 General Header Fields

There are a few header fields which have general applicability for both request and response messages, but which do not apply to the entity being transferred. These header fields apply only to the message being transmitted.

Table 1. Information Available in HTTP General Header

Header	Description
Cache-Control	The Cache-Control field is used to specify directives that **must** be obeyed by all caching mechanisms along the request or response chain. These directives typically override the default caching algorithms.
Connection	The Connection field allows the sender to specify options that are desired for that particular connection and **must not** be communicated by proxies over further connections.
Date	The date field represents the date and time at which the message originated.
Pragma	The Pragma field is used to include implementation specific directives that may apply to any recipient along the request/response chain. All pragma directives specify optional behavior from the viewpoint of the protocol. However, some systems may require that behavior be consistent with the directives.
Transfer-Encoding	The Transfer-Encoding field indicates what (if any) type of transformation has been applied to the message body in order to safely transfer it between the sender and the recipient. This differs from the Content-Encoding field in that the transfer coding is a property of the message, not of the entity.

Header	Description
Upgrade	The Upgrade field allows the client to specify what additional communication protocols it supports and would like to use if the server finds it appropriate to switch protocols.
Via	The Via field **must** be used by gateways and proxies to indicate the intermediate protocols and recipients between the user agent and the server on requests and between the origin server and the client on responses.

4.2.2 Request Header Fields

The request header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers with semantics equivalent to the parameter on a programming language method invocation.

Table 2. Information Available in HTTP Request Header

Header	Description
From	The From field contains the Internet e-mail address of the human user who controls the requesting user agent.
Accept	The Accept field can be used to specify those media types that are acceptable for the response. Accept headers can be used to indicate that the request is specifically limited to a small set of desired types as in the case of a request for an in-line image. If no Accept field is present, then it is assumed that the client accepts all media types.
Accept-Encoding	The Accept-Encoding field is used to indicate whether the response is encoded in any way, for example if it is a ".zip" or compressed file.
Accept-Language	The Accept-Language field restricts the set of natural languages that are preferred as a response to the request.
Accept-Charset	The Accept-Charset field can be used to indicate what character sets are acceptable for the response. If no Accept_Charset field is present, the default is that any character set is acceptable.
User-Agent	The User-Agent field contains information about the user agent (Web browser) originating the request. It is being used for statistical purposes, tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user-agent limitations.
Referer	The Referer field is ***very useful***. It allows the client to specify, for the server's benefit, the URI of the resource from which the Request-URI was obtained. This allows a server for example to generate lists of back-links to resources for interest, logging, or optimized caching. It also allows obsolete or mistyped links to be traced for maintenance.
Authorization	The Authorization field enables a user agent (Web browser) that wishes to authenticate itself with a server to do so by including an Authorization field with the request. The Authorization field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

Header	Description
Proxy-Authorization	The Proxy-Authorization field allows the client to identify itself to a proxy which requires authentication The Authorization field value consists of credentials containing the authentication information of the user agent for the proxy and/or realm of the requested resource.
If-Modified-Since	The If-Modified-Since field is being used to make the GET method conditional. The Web server returns the requested document only if it has been updated since the update supplied.
If-Match	The If-Match field is used with a method to make it conditional. A client (Web browser) that has one or more entities previously obtained from the resource can verify that one of those entities is current by including a list of their associated entity tags in the If-Match field.
If-Non-Match	The If-Match field is used with a method to make it conditional. A client (Web Browser) that has one or more entities previously obtained from the resource can verify that one of those entities is current by including a list of the associated entity tags in the If-Match field.
If-Range	The If-Range field is being used to make the GET methode conditional. If a Web client has a partial copy of an entity in its cache, it could copy the entire entity in its cache using the Range request header with a conditional GET.
Host	The Host field specifies the Internet host and port number of the resource being requested as obtained from the original URL given by the user or referring resource. A host definition without any trailing port information implies the default port for the service requested. (for example, 80 for an HTTP URL).
Range	The Range field specifies one or more sub-ranges of an entity to be copied with a conditional or unconditional GET
Max-Forwards	The Max-Forwards field may be used with the TRACE method to limit the number of proxies or gateways that can forward the request to the next inbound server.

4.2.3 Response Header Fields

The Response header fields allow the server to pass additional information about the response which cannot be placed in the status line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

Table 3. Information Available in HTTP Response Header

Header	Description
Age	The Age field conveys the sender's estimate of the amount of time since the response was generated at the origin server. A cached response is fresh if its age does not exceed its freshness lifetime.
Location	The Location field is used to redirect the recipient to a location other than the Request-URI for completion of the request or identification of a new resource.

Header	Description
Proxy-Authenticate	The Proxy-Authenticate field value consists of a challenge that indicates the authentication scheme and parameters applicable to the proxy for this Request-URI. The Proxy-Authenticate field **must** be included as part of a Proxy Authentication Required response.
Public	The Public field lists the set of methods supported by the server. Examples of methods are: GET, MGET, HEAD OPTIONS.
Retry-After	The Retry-After field can be used to indicate how long the service is expected to be unavailableto the requesting client.
Server	The Server field contains information about the software used by the origin server to handle the request. The field can contain multiple product tokens and comments identifying the server and any significant subproducts.
Vary	RFC2068 Chpt. 14.43 (pg 170)
Warning	The Warning field is used to carry additional information about the status of a response which may not be reflected by the response status code.
WWW-Authenticate	The WWW-Authenticate field MUST be included in Unauthorized response messages. The field value consists of at least one challenge that indicates the authentication scheme and parameters applicable to the Request-URI.

4.2.4 Entity Header Fields

Entity header fields define optional metainformation about the entity body or, if no body is present, about the resource identified by the request

Table 4. Information Available in HTTP Entity Header

Header	Description
Allow	The Allow field lists the set of methods that the requesting user can specify for this URL. (for example, GET, POST, HEAD).
Content-Base	The Content-Base field may be used to specify the base URI for resolving relative URLs within the entity. This header field is described as Base in RFC1808, which is expected to be revised.
Content-Encoding	The Content-Encoding field is used as a modifier to the media type. When present, its value indicates what additional content codings have been applied to the entity body, and thus what decoding mechanisms **must** be applied in order to obtain the media type referenced by the Content-Type header field. (for example, Content-Encoding: gzip).
Content-Language	The Content-Language field describes the natural language of the intended audience for the enclosed entity.
Content-Length	The Content-Length field indicates the size of the message body, in decimal number of octets, sent to the recipient or, in case of the HEAD method, the size of the entity body that would have sent had the request been a GET.

Header	Description
Content-Location	The Content-Location field may be used to supply the resource location for the entity enclosed in the message. Where a resource has multiple entities associated with it, and those entities actually have separate locations by which they might be individually accessed, the server should provide a Content-Location for the particular variant that is returned.
Content-MD5	The Content-MD5 field, as defined in RFC1864[23], is an MD5 digest of the entity body for the purpose of providing an end-to-end message integrity check (MIC) of the entity body. *Important:* ***MIC is good for detecting accidential modification of the entity body in transit, but is not proof against against malicious attacks!***
Content-Range	The Content-Range field is sent with a partial entity body to specify where in the full entity body the partial body should be inserted.
Content-Type	The Content-Type field indicates the media type of the entity body sent to the recipient or, in the case of the HEAD method, the media type that would have been sent had the request been a GET.
Expires	The Expires field gives the date after which the response should be considered stale. A stale cache entry may not normally be returned by a cache unless it is first validated with the origin server.
Last-Modified	The Last-Modified field indicates the date and time at which the origin server belives the variant was last modified.
ETag	The ETag field defines the entity tag for the associated entity. The entity tag may be used for comparison with other entities from the same resource.

4.3 Common Gateway Interface Scripts

CGI scripts do not have to be complex programs. Figure 7 shows a "Hello World" REXX CGI script that dynamically creates an HTML document to return to the Web browser.

```
/* A Message */
say "Content-type: text/html"
say""
say "<P>Hello World"
```

Figure 7. Simple REXX CGI Program

Figure 8 on page 40 shows a C program that performs a similar function.

```
#include <stdio.h>
#include <stdlib.h>
void  main() {
    printf("Content-type: text/html%c%c",10,10);
    printf("<H1> Hello World</H1>");
    return;
}
```

Figure 8. Simple C CGI Program

Note that for both REXX and C, you use standard functions to write output to the standard output stream (namely `say` and `printf`) to return data to the Web server.

`<H1>`, `</H1>`, and `<P>` are HTML tags. Tags are commands within an HTML document that describe its structure and formatting. Other formatting languages that use tags include the Standard Generalized Markup Language (SGML) on which HTML is based, and the generalized markup language (GML) used by IBM's Document Composition Facility and BookMaster. For a detailed introduction to HTML, see the following URL:

`http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-forms/overview.html`

When deciding the language in which to code your CGI programs, you have a choice of a wide variety of languages; there are the interpretive languages such as REXX and PERL, and languages that have to be compiled such as C and COBOL.

Most of the CGI documentation assumes that you will be using an interpretive language. These languages are ideal for the CGI environment. They offer powerful utilities for parsing incoming data, string handling to dynamically build HTML documents, and the ability to invoke other compiled programs when required. If you want to get something working quickly, then an interpretive language is probably the best way to start. If you are keen to optimize the performance of your Web server you may want to consider coding your CGI programs in a compiled language such as C or COBOL. Some C procedures available on the Web provide code to parse incoming forms data for you, so you do not have to code these utilities yourself. See:

`ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/cgi/`

4.3.1 Invoking Common Gateway Interface Scripts

How you tell your Web server to invoke a CGI script to process a URL, rather than retrieve an HTML document, depends upon the Web server you are using. NCSA, CERN, and Domino Go Webserver use server mapping directives, while for GoServe, this work is done by the filter program.

4.3.1.1 Server Mapping Directives

You use server script mapping directives to tell your Web server which URLs map to CGI scripts rather than HTML documents. The way in which this is done depends upon the Web server you are using:

- `ScriptAlias virtualpath actualpath`

  ScriptAlias is the NCSA implementation of server mapping directives. A ScriptAlias directive is placed in the configuration file for your Web server

specifying a script directory virtual path and an actual path for that directory. For instance, if you specify:

```
ScriptAlias /cgi-script/ /usr/local/etc/httpd/cgi-script
```

you are saying that any request specifying `cgi-script` in the transaction identifier part of its URL actually causes a program in directory `/usr/local/etc/httpd/cgi-script` to be executed. So if the following URL is received:

```
http:/baffin.sanjose.ibm.com/cgi-script/REXXCGI
```

then the Web server executes the following program:

```
/usr/local/etc/httpd/cgi-script/REXXCGI
```

You can have multiple ScriptAlias statements in your configuration file; your CGI scripts can then be in multiple directories.

- `Exec virtualpath actualpath`

  This is the CERN , and the Domino Go Webserver implementation of server mapping directives. It too is placed in the configuration file of the Web server. For the Domino Go Webserver for OS/2 Warp, this file is

  ```
  c:\tcpip\etc\httpd.cnf
  ```

  If, for example, you coded the following in your configuration file:

  ```
  Exec  \cgi-bin\*  C:\WWW\CGI-BIN\*
  ```

  and your Web server received the following URL,

  ```
  http://ladoga.sanjose.ibm.com/cgi-bin/REXXCGI
  ```

  the Web server invokes program `REXXCGI` in directory `C:\WWW\CGI-BIN`. You can have multiple `Exec` statements in your configuration file, so your CGI scripts can reside in more than one directory.

For more details on server mapping directives, see the following Domino Go documentation:

```
http://www.ics.raleigh.ibm.com/dominogowebserver/doc461.htm
```

4.3.1.2 Mapping Using GoServe

GoServe differs from the CERN and NCSA Web servers insofar as it does not use the CGI interface. Instead, it uses programs called *filters*, which look at the URL dynamically, and it is the filter itself that determines which filter program processes the incoming request. Figure 9 shows a fragment of REXX code that looks for the string `$CICSWEB` in variable `sel`, where `sel` is the REXX variable containing the URL. If `sel` contains `$CICSWEB`, the code invokes REXX program `cicsweb.80` to process the client request.

```
/*********************************************************************/
/* Modifications for CICS Web Interface.                             */
/*********************************************************************/
if translate(left(sel,8)) ='$CICSWEB' then do
  return cicsweb                          /* call the application */
end /* CICSWEB */
```

Figure 9. GoServe Mapping Code

Typically, you modify the REXX `GOFILTER.80` filter program provided as a sample with GoServe, to act as a router that looks at the incoming URL and invokes the appropriate filter to process the request.

See the following reference for details of GoServe:

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

4.3.2 Passing Data to Common Gateway Interface Scripts

There are three ways in which you can pass data to a CGI script:

- Environment variables
- Standard input stream
- Command line arguments

4.3.2.1 Environment Variables

A lot of information about the request received from the Web browser is passed to the CGI script in the form of environment variables. How these environment variables are retrieved depends upon the language in which the CGI script is written. Figure 10 on page 43 illustrates how the `getenv` function can be used by a CGI script written in C to retrieve the environment variables and return them to the Web browser.

See *Spinning the Web* for information about environment variables used by the CGI.

```
#include <stdio.h>
#ifndef NO_STDLIB_H
#include <stdlib.h>
#else
#endif
/*********************************************************************/
/*  This CGI script  retrieves all the environment variables         */
/*  passed to it by the Web Server, puts them into an HTML document, */
/*  and returns them to the Web browser.                             */
/*  Note that not all the environment variables will always have     */
/*  values associated with them, depending on what was sent by the   */
/*  Web browser.                                                     */
/*********************************************************************/
 main(int argc, char *argv[]) {

    printf("Content-type: text/html%c%c",10,10);
    printf("<H1> The CGI script was successfully invoked</H1>");
    printf("REQUEST_METHOD = ");
    printf(getenv("REQUEST_METHOD"));
    printf("\n");
    printf("<P>QUERY_STRING = ");
    printf(getenv("QUERY_STRING"));
    printf("\n");
    :
    :
    printf("<H1>HTTP Header Information</H1>");
    printf("<P>HTTP_USER_AGENT = ");
    printf(getenv("HTTP_USER_AGENT"));
    printf("\n");
    printf("<P>HTTP_ACCEPT = ");
    printf(getenv("HTTP_ACCEPT"));
    printf("\n");
    :
    :
}
```

Figure 10. Using the getenv C Function

Figure 11 on page 44 shows how a REXX CGI script on OS/2 could do the same.

```
/*********************************************************************/
/*  This REXX CGI script uses the "value" function to retrieve the */
/*  environment variables passed to it by the Web server, and      */
/*  place the values in an HTML document that is returned to the   */
/*  Web browser.                                                   */
/*********************************************************************/
say "Content-type: text/html"
say""
say '<H1>ENVIRONMENT VARIABLES PASSED TO THE CGI SCRIPT </H1>'
env = 'OS2ENVIRONMENT'

say '<P> QUERY_STRING = ' value('QUERY_STRING',,env)
      :
      :
say '<P> PRAGMA = ' value('HTTP_PRAGMA',,env)
```

Figure 11. Retrieve OS/2 Environment Variables Using REXX

There is a discussion of the use of the QUERY_STRING, CONTENT_TYPE, and CONTENT_LENGTH environment variables in 4.8.1, “Forms” on page 67.

Note that all the architected HTTP header information supplied by the Web browser (see 4.2, “Hypertext Transfer Protocol Header Information” on page 35) is passed as environmental variables, the names of which are the names of the header prefixed with HTTP_.

4.3.2.2 Standard Input Stream

If a request is made using the HTTP POST mechanism (see 4.8.1, “Forms” on page 67) then any input entered into a form by the user of the Web browser is passed to the CGI script in the script's standard input stream.

In this case, the MIME type of the data and its length are passed to the CGI script in the CONTENT_TYPE and CONTENT_LENGTH environment variables.

Retrieving and parsing the data is straightforward; the parsing functions provided by REXX make it an ideal language for this kind of operation, and C functions are readily available via FTP to do the same. See the following URL for an FTP directory containing C source for decoding forms input streams for both GET and POST:

ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/cgi/cgi-src

Most CERN and NCSA Web servers also provide the cgiparse utility (see 4.10, “Utility CGIPARSE” on page 76) to decode input streams. However, you should achieve better performance using the built-in facilities of your scripting language rather than an external program.

Figure 12 on page 45 shows one of the many ways in which data might be retrieved by using REXX. Note that some decoding has to be done, because HTTP replaces ""with "+" and uses "%" as an escape character to signal that the next two characters in the stream should be treated as the hexadecimal representation of the character. This example includes a function named packur to decode escape sequences. We also need this function in many of the later examples in this chapter. We do not repeat the source code in the later examples,

however. The program displays the input data it receives from the form that invoked it, both in its formatted and unformatted form.

Note that the CONTENT_LENGTH environment variable is used to establish the length of the data to be processed.

```
/* A Message */
say "Content-type: text/html"
say""

/* get the incoming data */
data = charin(,,value('CONTENT_LENGTH',,'OS2ENVIRONMENT'))
say 'Raw form data is "'data'<p>'
say 'Extracted variables are:<P>'

/* Parse input vars from form into var.x             */
/* set VAR.x variables from the incoming data stream */
VAR.=''                                 /* null string unless set  */
data=data'&'                            /* set end condition       */
do until data=''                        /* each value              */
  parse var data assign '&' data        /* split off name=value    */
  parse var assign name '=' value       /* separate                */
  value=translate(value, ' ', '2b090a0d'x) /* handle '+' tabs & CRLF*/
  name=translate(packur(name))          /* caseless name           */
  var.name=packur(value)                /* set, after URL decoding */
  say ' var.'name 'is "'var.name'"<p>'
end
return ''

/*********************************************************************/
/* Packur:  This procedure takes an input string, and converts       */
/* characters encoded as escape sequences (e.g. %xx) back to the     */
/* character that they represent.                                    */
/*********************************************************************/
packur: procedure
  parse arg outstring '%' rest
  do while length(rest) > 1
    outstring = outstring||x2c(substr(rest,1,2))      /* decode value */
    rest = substr(rest,3)
    parse var rest next '%' rest
    outstring = outstring||next
  end
  return outstring
```

Figure 12. REXX Program to Decode Forms Input

4.3.2.3 Command Line Input

This is the least common means of passing data from the Web server to the CGI script. It is used only if the Web server cannot find an "=" in an inbound query string, so the incoming data is not a form. In this case, the Web server splits the string into component words, using the "+" symbol as the delimiter (the HTTP protocol cannot handle embedded blanks in URL strings, so the rule is that blanks are replaced by "+" symbols).

If data is passed to the CGI script in this format, then the first argument (arg0) passed to the CGI script is the script command name.

4.4 Internet Connection Application Programming Interface ICAPI

On the OS/390 internet server (Internet Connection Secure Server on OS/390 Rel. 1.3, and Lotus Domino Go Webserver for OS/390 on OS/390 Rel. 2.4) there is an alternativ way to CGI, to access external programs which interface with the Web Server. This alternativ to CGI is called Internet Connection Application Programming Interface (ICAPI). ICAPI, other than CGI, offers a much faster access path to mainframe resources for Web access.

The ICAPI intreface allows easily to expand the OS/390 Web server's base functions with own customized processing routines. This can be done by specifying **directives**. Directives make the OS/390 Web server to call specified application functions in a program at various points in its request processing cycle. These directives need to be specified in the configuration file of the OS/390 Web server. The name of th configuration file is httpd.conf.

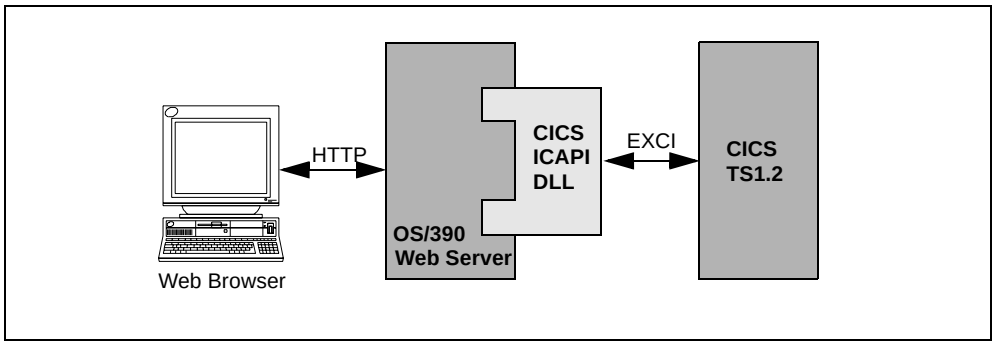


Figure 13. CICS ICAPI Overview

4.4.1 The Service Directive

CICS Transaction Server 1.2 uses only one of the directives offered by ICAPI. The **service** directive is used to specify a customized application function it wants the server to call during the service step. The format of this directive is:

```
Service request-template /path/file:function_name [Server-IP address or hostname]
```

`/path/file`
is the fully qualified name of a compiled program.

`function_name`
is the name given to the function within the program specified in /path/file.

`request_template`
is a template that determines if the specified application function is called.

If we want to use the OS/390 Web server in combination with the CICS Web Interface (CWI), which is explained in Chapter 5.1.4, "CICS Web Interface" on page 80, CICS Transaction Server 1.2 requires a service directive in the httpd.conf file of the following format:

```
Service /applid/*/home/dfhwbapi.so:DFHService
```

In aboves example, Service is the name of the directive. The request_template corresponds to the applid of a CICS region. /path/file corresponds with /*/home/dfhwbapi.so and is the name of a CICS supplied program. Finally, the function_name DFHService is the name of a function within this CICS program.

This directive is nothing else but a mapping ruleto indicate when dfhwbapi.so has to be invoked. If the name of the applid in the incoming URL is the same as the name specified in the service directive, control is passed to dfhwbapi.so. /applid/ is the name of the applid in the CICS region where we want to execute the user program

4.5 Java

The Java language was developed by Sun Microsystems in 1991 as part of a research project to develop electronic devices like television sets, VCRs or coffee machines. The development goals for Java at that time were to have a programming language that is portable (to be used in any electronic consumer device), easy, fast, and small. Although Java has been used in several small projects, it did not make its breakthrough until HotJava was invented. HotJava was developed in 1994 as a vehicle for downloading and running Java applets and to show how to develop Java applications. By today, HotJava has been more or less replaced by modern Web browsers like the latest IBM WebExplorer, Netscapes Navigator 2.00+, or Microsofts Internet Explorer 3.00+. These Web browsers give the the Web users the possibility to download and run Java applets across networks and platforms.

4.5.1 What Is Java?

Java is an interpreted and object-oriented language similar to C++. It can be used to build programs that are platform independent in both source and object form. At the source level, Java's primitive data types have consistent sizes across all development platforms and Java's foundation class libraries make it easy to write code that can be moved from platform to platform without the need to implement platform specifics. These classes include graphical user interface functions, input/output functions, and network communications. The binary compatibility is given through platform independent byte code which is produced with a Java compiler. How does that exactly work?

Where traditional compilers like a C or C++ compiler translates everything into processor-dependent machine code or processor instructions, the Java byte code compiler produces universal intermediate byte code, which is executed at run time by a platform-specific byte code interpreter. The interpreter also checks the byte code at execution time to ensure its validity and safety to the machine environment. The isolation which the Java interpreter provides, coupled with the platform-specific Java run time system, builds the platform independent Java Virtual Machine (JVM) environment. Netscape Navigator 2.00+, and Microsoft Internet Explorer 3.00+, support the JVM environment.

The Java language can be used to construct Java applets and Java applications. Both are used in the CICS Gateway for Java.

4.5.2 Java Applets

Java applets, the small Java programs that are downloaded from the Web and executed by a Web browser or a network computer, typically perform the type of operations that client code would perform in a client/server architecture. An applet edits input, controls the screen, and communicates transactions to a server, which performs the data or database operations.

The "Hello World applet" in Figure 14 on page 48 shows the basic structure of a Java applet. The import line at the top is very similar to the #include statement of a C/C++ program. It enables access to the Java foundation classes. " The method that displays the applets content in graphics mode to the screen is "paint", in this example the string "Hello World".

```
import java.awt.Graphics;
import java.awt.Font;
public class HelloWorld extends java.applet.Applet
{
Font font = new Font("TimesRoman", Font.Bold, 32);
    public void paint(Graphics g)
    {
        g.setFont(font);
        g.drawString("Hello World", 5, 40);
    }
}
```

Figure 14. "Hello World" applet

In order to be able to run this applet, you must first compile it with the Java compiler.

```
javac HelloWorld.java
```

After the compile, you should have the file HelloWorld.class, which now can be implemented into a HTML page.

The invocation of applets occurs through the use of the HTML applet tag. This tag is used in HTML pages to indicate when applets are to have control and to specify the display area to be used by the applet. When a Java-enabled server is downloading a page and encounters this tag, it also downloads the applet byte code in the same way it downloads an image which is referenced by an HTML image tag. The Java-enabled browser then interprets and executes the applet byte code.

```
<HTML>
<HEAD>
<TITLE>Good morning to everybody</TITLE>
</HEAD>
<BODY>
<P>Following is our Java applet:
<APPLET CODE="HelloWorld.class" WIDTH=100 HEIGHT=20>
</APPLET>
</BODY>
</HTML>
```

Figure 15. HTML with Applet Tag

The downloading of applets should not have a significant performance impact on the response time to end users, since the applets are typically not very big. In fact applets, by performing processing on the browser or network computer, can improve overall browser performance by eliminating iterations with the Web server. As with images, applets are cached in Web browsers. This minimizes the frequency of applet downloading.

4.5.3 Java Applications

Java applications are general programs written in Java and executed locally on a computer. Java applications don't require a Web browser to run and allow programming operations in addition to those used in applets, which can otherwise make the code platform-dependent. It can access local files, it can create and accept general network connections, and it can call native C/C++ functions in machine-specific libraries.

Java applications can use the CICS-provided Java classes to do transaction processing in CICS systems. They can use the JavaGateway class to establish two kinds of connection:

- **A network gateway connection** is a connection to a CICS Gateway for Java. When the connection has been established, an application can use the ECIRequest or the EPIRequest class or both, to do transaction processing.
- **A local gateway connection** is a connection that, despite its name, does not use a CICS Gateway for Java. The facilities available depend on where the application is running:
  - If the application is running on a workstation with a CICS client, the connection is to the CICS client. The application can use the ECIRequest, or the EPIRequest class, or both, to do transaction processing.
  - If the application is running on a workstation without a CICS client, a local gateway connection cannot be established.
  - If the application is running on an MVS system, the connection is to a CICS pseudo-client that uses EXCI to communicate with CICS systems. The properties of the pseudo-client (connected CICS system, for example) are determined by environment variables set in the MVS system. The application can use the ECIRequest (not the EPIRequest) class to do transaction processing.

4.5.4 Java Beans

Java Beans is the component model for Java. The intention of Java Beans is to define a model, thus enabling suppliers to create and ship Java components that can be composed into applications by end users. For more details on Java Beans, see:

`http://java.sun.com/docs/books/tutorial/beans/whatis/software-components.html`

or

`http://java.sun.com/beans/related.html`

4.6 Java Security

This section describes the security issues you need to be aware of when developing a Java-based NC solution. Java security features are currently in a state of change, and vary depending upon browsers and JVM implementations. The information here should enable you to write a Java program that will not be broken by security restrictions when you deploy it.

4.6.1 Java Security Features

In general, Java applets that are loaded from a network are deemed untrusted and are restricted in the actions they can perform. Some of the things an untrusted applet cannot do are listed here:

- It cannot make network connections to hosts other than the originating host.
- It cannot read and write files on local disks.
- It cannot start external programs and processes.
- It cannot load libraries and define native methods.

These restrictions allow a user to download an applet with reasonable assurance that it cannot do harmful things to their system. It is up to the implementation of the Java Virtual Machine (JVM) in the browser to ensure that these restrictions are policed. Security bugs in JVM implementations have been found and Javasoft maintain a chronology page, `http://www.javasoft.com`, on their Web site to record the status of these bugs. Since its introduction, the Java technology has evolved in the public eye and when security bugs are found, fixes have quickly followed.

Web browsers that support Java 1.0 applets are very strict about enforcing the security rules and the term *sandbox* has been used to describe the restrictive environment that the applet runs in. This is not a problem for small applets that enhance the look and interface of Web pages, and do not provide much complex functionality. Applications written in Java do not have these same restrictions and can access the system in the same ways as traditional compiled programs. This is because applications must be installed locally on a computer and run from the local disk. It is the responsibilty of the person installing the application to know what it is and that it can be trusted. This approach loses the advantages of dynamically downloadable applets, where no code needs to be installed on the users‘machines. Java 1.1 addresses this issue by allowing applet class files to be digitally signed by the developer. This allows the end user to determine the degree of trust that the applet can be afforded. When the trust is established, the applet can be allowed to perform actions that an untrusted applet could not do.

4.6.2 Leaving the Sandbox

The latest versions of the popular Web browsers, Microsoft Internet Explorer Version 4, Netscape Communicator Version 4, and HotJava Version 1.1 provide mechanisms for the user to allow Java applets access beyond the sandbox. These are based on digital signature and certificate technology and interaction with the user of the browser. When a Java applet has been digitally signed, the user can allocate security settings based on the signature. The applet can then do things that would otherwise not be allowed if it had no signature. See 4.6.6, “Digital Certificates” on page 55 for details on how using digital certificate technology allows the user to make these security choices safely.

Once a mechanism for trusting a Java applet is used, the developer is free to write code for any purpose. You have the full power of the Java language and can write sophisticated client programs without any restrictions. These then use the dynamic downloading capabilities of browsers, removing the need to administer client platforms and manually install software. Within an Intranet environment, the applet trust should be implicit, since the code has been developed by the company and the intranet is secure. The use of external trusted third party

Certification Authorities as described in 4.6.6, “Digital Certificates” on page 55 would not be necessary.

The only drawback at the current time is the mechanisms used by different browsers to establish whether or not an applet is trusted. Each of the popular browsers uses a different mechanism, and Netscape Communicator requires the use of special API calls in the Java applet code to request the privileges to perform insecure actions. The certificate technology required to sign the code is also different for each browser. Applet code that has been signed for one browser will not work with the others. You are required to provide a separate copy of the signed code for each possible browser.

Sun Microsystems have evolved the sandbox model in their architecture for Java 1.2 to include fine-grained access control. The details are in the Java Security Architecture (JDK 1.2) document available from Sun Microsystems. The new APIs have a purpose similiar to that of the Netscape Communicator Capabilities API described in 4.6.3, “The Netscape Capabilities API” on page 51. When Netscape Communicator supports Java 1.2, developers should no longer need to use the Capabilities API. Other browsers that claim support for Java 1.2 will also need to provide the security APIs that will be part of the core Java API. This will solve the problem of requiring specific code for particular browsers. Java 1.2 should also provide a standard for signing JAR archive files that will be common to Java browsers. This will allow for a single JAR files to be signed and then used in any browser.

4.6.3 The Netscape Capabilities API

A signed Java applet that runs in Netscape Communicator is not automatically allowed to access system resources. The developer must use calls to the Capabilities API to enable privileges before the actions can be taken. There is a **PrivilegeManager** class that controls access to a set of system targets. For example, in order to successfully make a network connection, the following code must be used:

```
PrivilegeManager.enablePrivilege("UniversalConnect");
```

The complete set of targets covers all the things an applet may aim to do, and groups of targets allow for more general access. For example,

```
PrivilegeManager.enablePrivilege("TerminalEmulator");
```

includes the targets *UniversalLinkAccess*, *UniversalPropertyRead*, *UniversalListen*, *UniversalAccept* and *UniversalConnect*. The Netscape developer web site http://www.javauniverse.com/Developer/97/06/SigningApplets/index.html contains details on using the API, including a complete list of all the targets.

There is also an article at the JavaUniverse Web site by Joseph Bowbeer, entitled "Signing Applets for Internet Explorer and Netscape Navigator" at http://www.javauniverse.com/Developer/97/06/SigningApplets/index.html. It contains useful code examples on using the API.

4.6.3.1 User Intervention and the Security Information Panel

When an applet makes an *enablePrivilege* call for the first time, the user must decide how to process the request. Netscape Communicator displays a panel like Figure 16, “Netscape Security Request” on page 52

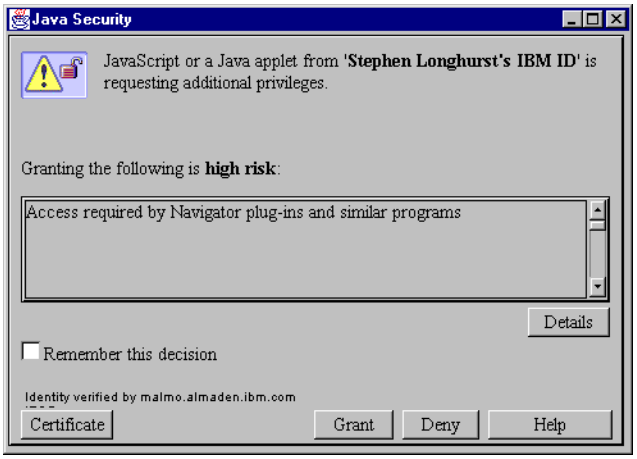


Figure 16. Netscape Security Request

The name of the certificate is displayed, "Stephen Longhurst's IBM ID" in Figure 16 on page 52, and a short description of the target being requested. The full certificate can be displayed by clicking on the Certificate button. If the certificate was not issued by a recognized authority then the applet is automatically denied the requested privilege with no user intervention.

The user can grant or deny the privilege, and set the checkbox so that the panel is not displayed every time the applet enables the privilege. You can check and alter the privileges afforded to a certificate by using the using the "Security Info" option any time.

The advantage of this method of setting the privileges is that the user does not need to know in advance what an applet needs to do to run properly. During the execution, the messages are displayed as required. We recommend issuing *enablePrivilege* calls for each target your applet requires access to when execution first starts. This gets the user dialogs out of the way at the beginning, rather than interrupting the user during normal operation.

4.6.3.2 Principles

The principles in the Capabilities API refer to who is allowed to do certain actions, and the targets refer to what they want to do. Section 4.6.3 describes some of the targets. The principle of an applet is normally associated with its signature, hence only signed applets can make use of the Capabilities API. It is the name of the signature that is listed in the `Java/JavaScipt` section of the "Security Info" panel. For developement purposes, having to sign the applet continuously for testing is not very productive. You can get Netscape Communicator to recognize the applet's code base as a principle when evaluating the privileges by adding the following line to the preferences file:

```
user_pref("signed.applets.codebase_principal_support", true);
```

This allows you to write code that uses the Capabilities API, but does not need to be signed.

4.6.4 Microsoft Internet Explorer Security Zone System

Microsoft Internet Explorer 4 introduces the concept of dividing the Web into different zones. You can assign different levels of security to each zone, trusting

each at a different level. For example, you can set the Intranet zone to low security (high trust) but the Internet zone to a high security level (low trust). Figure 17, “Microsoft Internet Explorer Security” on page 53 is the security settings panel in Microsoft Internet Explorer Version 4. You can see the Internet zone has a High security setting by default.

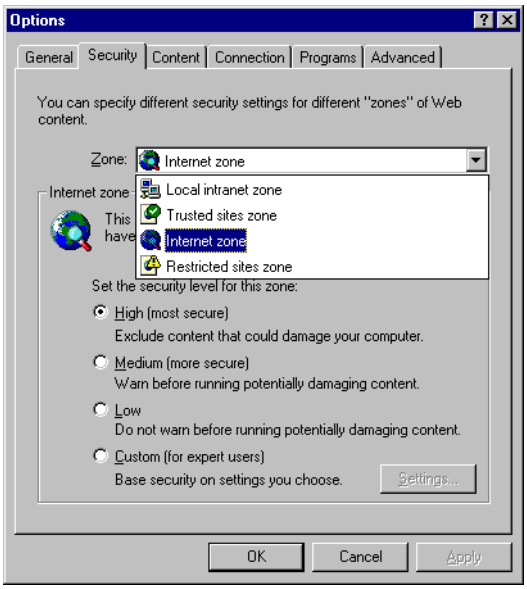


Figure 17. Microsoft Internet Explorer Security

Java applets are allowed levels of access and capabilities based on the zone from which they originate. By signing an applet, the developer can also specify the security level the applet needs to run at. If the applet is loaded from a higher security zone, the user needs to decide if the applet will be allowed to do what it requests.

The capabilities that a Java applet can be granted are defined in High, Medium, and Low settings. The High setting is equivalent to the sandbox-like environment. The Medium setting adds the capabilitity to access a scratch pad. The scratch pad is a secure area, proprietary to Microsoft Internet Explorer, which allows applets to store information on the client machine. It is accessed with Java APIs that are provided with the Microsoft Java Developers Kit. The Low setting allows applets complete access to the system.

Microsoft claims this approach allows fine-grained control over what a Java applet is allowed to do. If you discount the Medium setting because it is the same as the High setting with one extra capability, you have a either a sandbox or complete access to everything. As a user, if you allow a Java applet to run with Low security, then it has complete access to your system. This is unlike the Netscape model that allows you to control exactly which resources an applet can access.

4.6.5 The HotJava Security Model

The HotJava browser has quite a flexible approach to security from a user‘s point of view. A user can exert fine control over the things that an applet can and cannot do, based on the site it comes from or the certificate it is signed with. The

security dialogs allow for defaults as well as customised settings, and you can group sites and certificates together for easy configuration. This greater flexibility makes it easier for somebody to unwittingly allow an untrusted applet to perform undesired operations. Applets that are not signed can be allowed privileges that in the other browsers are available only to signed applets.

Figure 18, “HotJava Basic Security” on page 54 shows the basic security dialog in HotJava. You can set the options for both signed and unsiged applets. The advanced options, shown in Figure 19, “HotJava Advanced Security” on page 55 allow you to configure specific permissions granted to applets. This is very comphrensive and allows control over things like specific sites the applet can or cannot connect to, exact directories on your disk that the applet can or cannot write to, and the system properties the applet is allowed to access.

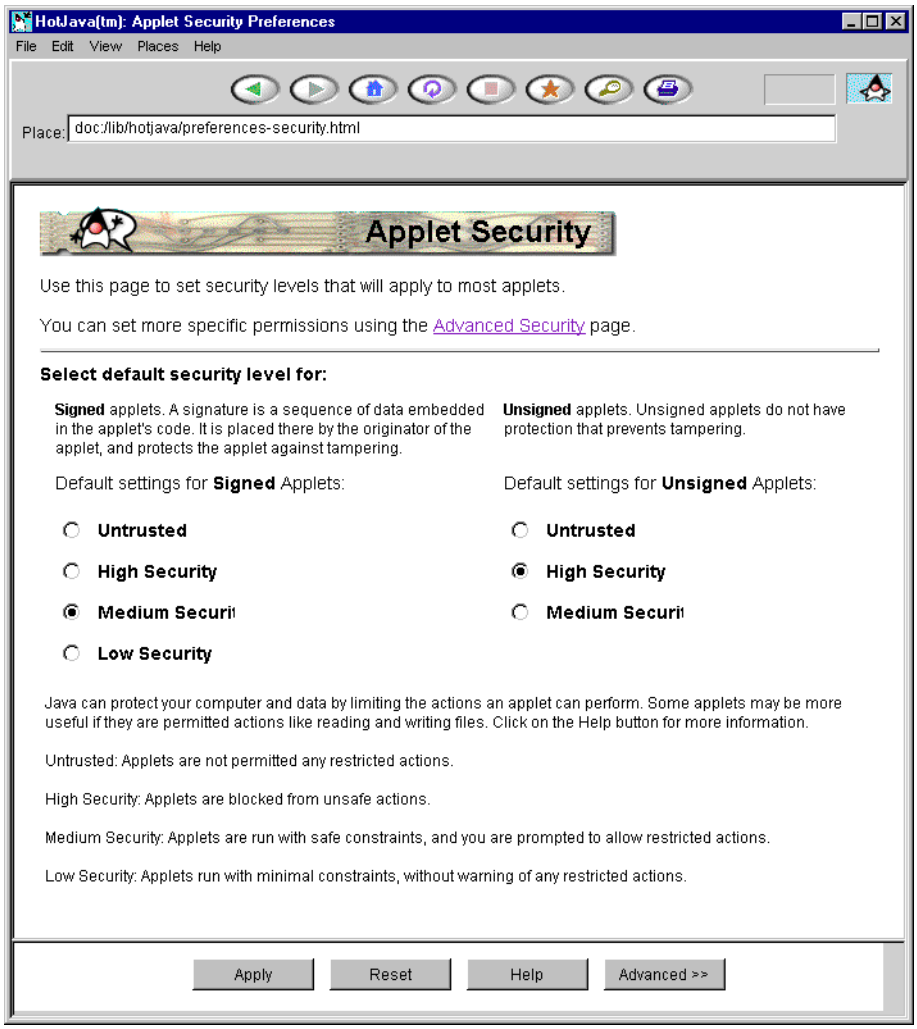


Figure 18. HotJava Basic Security

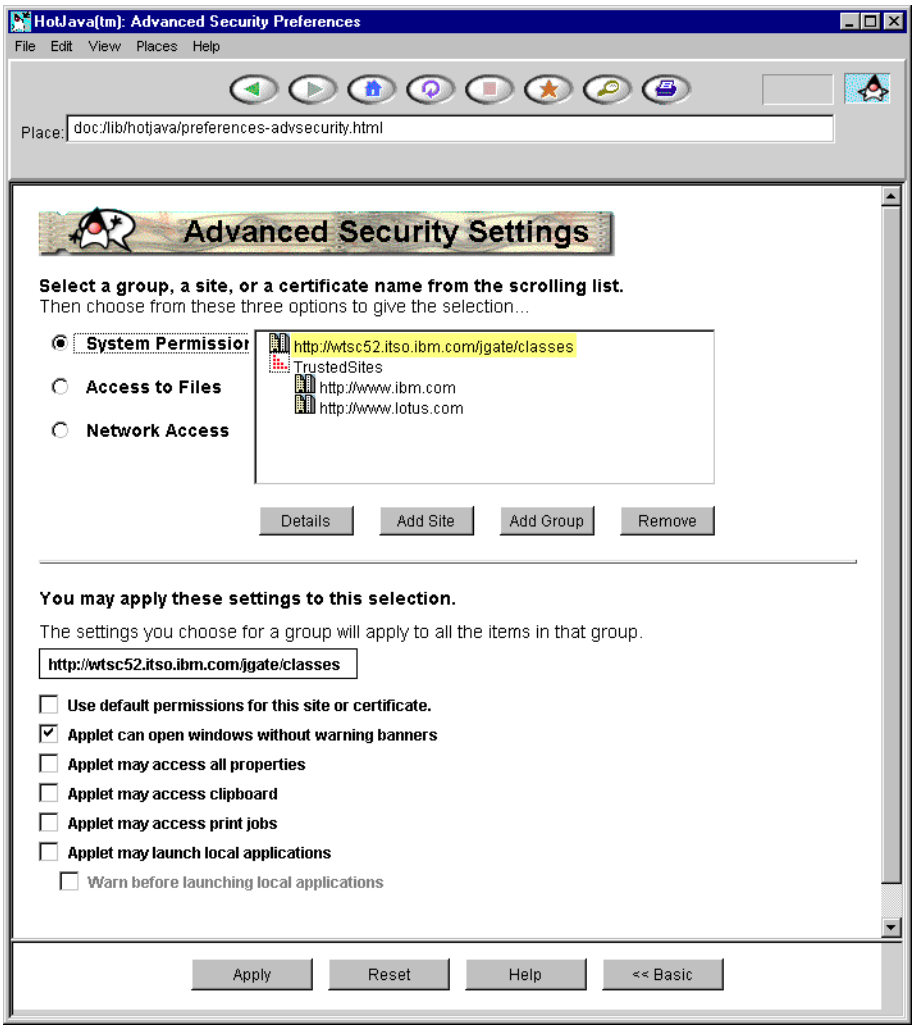


Figure 19. HotJava Advanced Security

HotJava is relatively new and less sophisticated than the other browsers. One thing is missing, which is available with Microsoft Internet Explorer and Netscape Communicator: a centralized administration point. By using administration products from Microsoft or Netscape, end users browsers can be configured by a single administrator. The configurations can then be locked so that users cannot alter them. This allows administrators to decide on a security policy that all browsers must follow, and prevent the end users from circumventing this. In a company Intranet environment, enforcing a security policy may be very important.

4.6.6 Digital Certificates

A digital certificate is a data structure that contains three pieces of information: a name, a Public Key and a Digital Signature computed over the other two. The certificate is signed by a trusted third party called a *Certification Authority* (CA). It is the job of the CA to verifiy that the name information is correct. If you trust the CA, then digital certificates provide a safe method for distributing public keys via an electronic medium.

4.6.6.1 How to use a Digital Certificate

Your copy of Netscape Communicator or Microsoft Internet Explorer comes loaded with the digital certificates of trusted CA's such as Verisign http://www.verisign.com, and the IBM World Registry http://www.internet.ibm.com/commercepoint/registry/index.html. Figure 20 on page 56 shows the Netscape Communicator security info dialog, displaying a list of the CA certificates. If you create your own CA for Intranet use, then your browsers must be loaded with your CA certificate. This function is provided by the Certificate Server, and is the only manual configuration step that the browser users must do before running applets signed with your developer certificates. Figure 21 on page 57 shows a Netscape Communicator browser importing a new CA certificate.

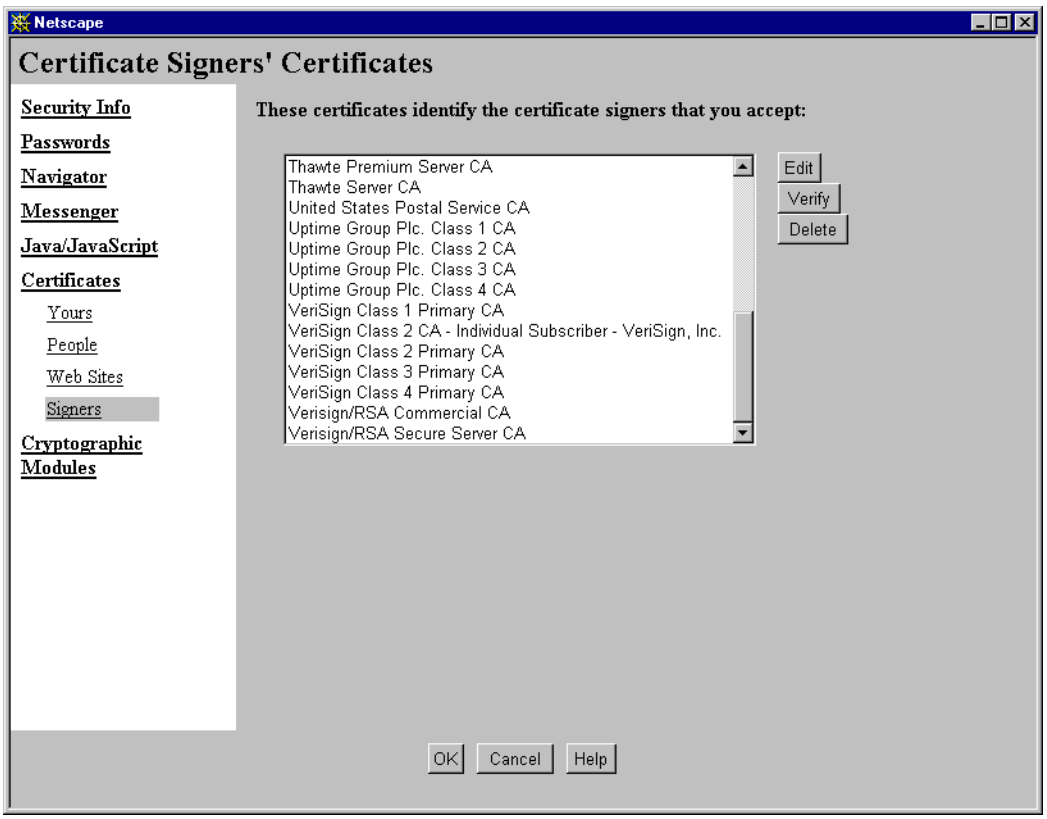


Figure 20. Netscape Certificates Dialog

Figure 21. Netscape Import CA Certificates

4.6.6.2 Why Sign Java Applets?

In order for downloaded Java code to have access to a system beyond the sandbox, you must explicitly grant it privileges. In order to make the decision as to whether to grant the access or not, certain conditions must be met.

Authenticity

You must be able to authenticate who developed the applet, and that you trust that person. With off-the-shelf software, the packaging usually identifies the software publishers explicitly, and other physical identifiers like holograms and certificates of authenticity allow you to trust the program you install on your machine. When you download an applet into your Web browser, you do not get the same guarantees about where the software has come from, or who developed it.

When Java code is digitally signed, you have reliable information about who developed the code. You can then make the decision about what you are going to allow the program to do. When the Web server employs secure sockets, you can be assured of where the code is being downloaded from.

Integrity
You need to be sure of the integrity of the code that you download. If it has been altered in any way during transmission, you cannot trust it. Encryption and digital signatures ensure integrity.

Accountability
When you receive code that is signed by a particular developer or organization, you can be sure that they cannot deny it is their code. Only they should have knowledge of their secret key, which is used to create the signature. If something does go wrong, you know who is accountable for the mistakes.

4.6.6.3 Obtaining a Digital Certificate

The *javakey* program and Microsofts code signing technology allow you to create your own certificates for signing Java code. The Java security Web pages at Sun `http://java.sun.com/products/jdk/1.1/docs/guide/security/index.html` describe how to create certificates using *javakey* and “Microsoft Authenticode Technology” on page 61 shows how to use the *makecert* program to create a Microsoft code signing certificate.

Obtaining a Netscape code signing certificate is more difficult. You can purchase one from a commercial CA such as Verisign Inc. `http://www.verisign.com` or Thawte Consulting `http://www.thawte.com`. The average cost is about $20 per year. Verisign calls the certificates enabled for code signing Class 2 or Class 3 certificates. Class 2 are for personal use while Class 3 are corporate certificates and cost significantly more. At the time of writing, Verisign issue these certificates only to US and Canadian residents. Netscape maintains a Web page ehttps://certs.netscape.com of companies that issue certificates to clients, but they are limited today to Belgium, Luxembourg, Brazil, Spain, and South America. Thawte Consulting offers services in more countries, but not for code signing certificates in every country. The lack of services in countries other than the U.S. is due to the difficulty in verifying individual identities. The U.S. Social Security number provides a consistent method to uniquely identify a U.S. resident, while not all other countries have a similiar mechanism. Verisign and the other companies also offer Microsoft code signing certificates if you need to publish software for the Internet.

If you do not want to pay money for a Verisign certificate, or you live in a country where certificates are not available, you can download the Netscape Certificate Server evaluation copy. This allows you to set yourself up as a certificate authority, within an Intranet environment, and issue yourself and others, developer‘s certificates. These certificates, like the Microsoft developer‘s test certificates, will not be trusted on the public Internet. Using the Certificate Server is a good way to learn about the technology and become familiar with the processes of obtaining and using digital certificates. It is quite simple to install and set up, and comes with comprehensive instructions. There is also a patch available from Netscape that allows the Certificate Server to issue certificates to the Microsoft Internet Explorer browser. A detailed description of the Netscape Certificate Server is beyond the scope of this chapter, but it is worth looking at for evaluation.

4.6.6.4 Creating Signed Java Applets

This subsection describes the steps required to build a signed archive file for Netscape Communicator, Microsoft Internet Explorer, and HotJava. Signing the

applet class files and placing them in an archive enables browsers to give aditional privileges over unsiged applets.

4.6.6.5 The Netscape Tools

Two tools for signing Java code are available from Netscape: JAR Packager and zigbert. JAR Packager is a Java applet that is run within Netscape Communicator and has a graphical user interface. Zigbert is a command line tool, available for Windows NT. Both can be downloaded free of charge from the Netscape developer site `http://developer.netscape.com`. JAR Packager proved slow and unable to handle large numbers of files. In order to create a JAR file which included all the VisualAge support classes and our applet, zigbert had to be used.

Follow these steps to create a signed JAR file using zigbert. The steps assume that zigbert has been downloaded and installed on your machine, and the directory is in the PATH:

1. You must have a certificate installed in your Netscape Communicator that is enabled for object signing. See “Obtaining a Digital Certificate” on page 58 for ways to obtain this certificate. Make a note of the name by which the certificate is referred to when you installed into the Netscape Communicator database.
2. Place all the files that compose your applet into an empty directory, making sure the directory structure is preserved. If you are using VisualAge, use the export project option and specify the directory, but do not select the export JAR file option.

   To sign all the files, issue the following command (all on one line):

   ```
   zigbert -d "c:\program files\netscape\users\default" -k \
   "Stephen Longhurst's IBM ID" c:\build
   ```

   This assumes that the netscape certificate database files are in the directory `c:\program files\netscape\users\default`. This is usually the case. The files are named cert*.db and key*.db. The name of the signing certificate is "Stephen Longhurst's IBM ID". You can check the names of your certificates by using the *Security Info* panel in Netscape Communicator and looking under the *Certificates->Yours* section. The directory where all the files have been expanded to is `c:\build`.

   You will be prompted for the password that protects the certificate you specify. After the command has completed, a new directory called META-INF is created under the top level (c:\build). This directory contains the manifest file for the JAR archive and the Netscape-specific signing information files.
3. Use the zip command that is supplied with zigbert to create the JAR file from the directory. Change to the top level directory (c:\build) and use the command:

   ```
   zip -r ns4applet.jar .
   ```

   A file is created called ns4applet.jar. This is the JAR file that is uploaded to the Web server and is referenced in the *archive* tag of the HTML page. You can use any name you like for the output file.

Figure 22 and Figure 23 on page 60 shows sample script files that were used to generate signed applet archive files:

```
REM makens4.bat
REM Script to digitally sign a directory structure of files
REM and then package them up into a JAR file.
REM Use on Windows NT
REM
REM Parameter 1 = Directory to package up
REM Parameter 2 = Name of output file
REM
REM Example Usage : makens4 g:\build ns4applet.jar
REM

SET ZIGBERT_DIR=c:\JARPackager\zigbert
SET CERTDB_DIR="c:\program files\netscape\users\default"
SET CERTIFICATE="Stephen Longhurst's IBM ID"

%ZIGBERT_DIR%\zigbert -d %CERTDB_DIR% -k %CERTIFICATE% %1
cd %1
%ZIGBERT_DIR%\zip -r %2 .
```

Figure 22. Create a JAR File Signed for Netscape Communicator

```
REM makeie.bat
REM Script to digitally sign a directory structure of files
REM and then package them up into a CAB file
REM Use on Windows NT
REM
REM Parameter 1 = Directory to package up
REM Parameter 2 = Name of output file
REM
REM Example Usage : makeie g:\build iecashier.cab
REM

SET SDK_DIR=c:\sdk-java.20\bin
SET CERT="g:\ie_certs\itsokey.spc"
SET NAME=ItsoKey
cd %1
%SDK_DIR%\cabarc -r -p -s 6144 n %2 Residency\* Domain\* COM\* ibm\*
%SDK_DIR%\signcode -j javasign.dll -jp low -spc %CERT% -k %NAME% %2
%SDK_DIR%\chkjava %2
```

Figure 23. Create a CAB File Signed for Microsoft Internet Explorer

4.6.6.6 The Sun JDK Tools

The Sun Java Developers Kit (JDK) Version 1.1 includes the *javakey* tool that allows you to create, display, and save certificates. It is also used to sign JAR files for use with HotJava. Javakey manages a database of entities. These entities are either identities or signers and the user or administrator can declare certain entities to be trusted.

The Sun Microsystems Web site contains comprehensive instructions on using the *javkey* tool as well as a tutorial on signing applets. The page is entitled "Security and Signed Applets"•.

For the purposes of this project, actually signing a JAR file for use with HotJava was not necessary. HotJava can be configured to allow unsigned applets privileges beyond the sandbox. We used this mechanism when testing the applet with HotJava. You need to create a separate JAR file for use with HotJava because one signed with Netscape's tools will fail to load properly. To create the JAR file, use the *jar* tool in the directory where your class files are expanded.

```
jar -cvf output.jar *
```

4.6.6.7 Microsoft Authenticode Technology

You can use Microsoft's tools to create digitally signed CAB files, giving your applets access to system resources when running in Microsoft Internet Explorer. You need to have the Microsoft Java Software Developers Kit (SDK). the currently available version is Version 2.0 beta 2. The SDK provides a tool called *makecert* that allows you to generate a test software publishers certificate to sign the CAB file with. The Java SDK is available from the Microsoft Web site.

The followings steps are detailed on the Microsoft Web site in a document entitled "Signing a Cabinet File with Java Privileges using Signcode". It is assumed that the Java SDK is installed on your machine with the bin directory in the PATH.

1. Create a certificate with the makecert program using the following command:

   ```
   makecert -sk DeveloperKey -n "CN=Company Development" TestCert.cer
   ```

2. Use the *cert2spc* program to turn the certificate into a test software publishers certificate:

   ```
   cert2spc TestCert.cer TestPublish.spc
   ```

3. Extract all your Java applet class files into an empty directory. This step is equivalent to Step 2 in "The Netscape Tools" on page 59. Use the cabarc tool to create a CAB file containing all the files:

   ```
   cabarc -r -p -s 6144 n output.cab *
   ```

4. Use the sign code program to sign the CAB file with your software publishers certificate (this is all one command):

   ```
   signcode -j javasign.dll -jp low -spc TestPublish.spc -k DeveloperKey
   output.cab
   ```

   The -jp options specifies the security level for which the CAB file is signed. If your code needs to do anything beyond the sandbox other than access the scratch pad, it must be signed with low security.

4.6.7 Java Application Security

Enabling SSL security on your Web server (see "Secure Sockets Layer" on page 24) enables Java code to be downloaded securely, providing code integrity. Digital certificates provide mechanisms for authenticity and accountability, allowing users to trust applets. Neither of these mechanisms automatically provides security for any communication that the Java code then independently performs. It is up to the applet code that you write to provide security for these connections. Currently, there is no easy way to do this for the CICS Gateway for Java. Future product enhancements will include providing encryption, in the form of SSL, for the CICS Gateway for Java. For the Intranet, this is not an inhibitor to using the technology. Use on the Internet will be restricted to applications that do not require confidentiality of information, until encryption is supported.

4.6.8 Security Features in Java 1.2

The Java 1.1 platform does not provide concrete implementations of some key security features. Certain certificate formats and algorithms will be introduced only with Java 1.2. The certificate management infrastructure includes support for X509v3 certificates. A new permission-based security mechanism is also proposed, similiar to the Netscape Capabilities API. This will allow common code that requests access outside the sandbox to be written. It will be compatible with all browsers and Java environments. The problem today is the need for browser-specific code, the Netscape Capabilities API, to be written. The support for X509v3 certificates will remove the need to sign applet code for specific browsers. More information on security and the proposed specification for Java 1.2 is available at Sun's Web site.

4.6.9 Summary

This section provides a short summary of the security issues you should be aware of when developing a Java-based network computing application.

4.6.9.1 Writing Full Function Java Applets

You need to be aware of how Web browsers enforce security on Java applets if you want to write code that needs access to the system beyond that allowed by the sandbox model. Different browsers use different mechanisms to check if code can be trusted:

- For use with Netscape Communicator, your code must include calls to the Capabilities API. You create a JAR file that must be digitally signed with a code-signing certificate compatible with the Netscape tools.
- For use with HotJava, you must inform the browser users of the requirements of your applet. They then have to configure the browser to allow the applet the access it requires. Your code may or may not be digitally signed, but if it is, you must create a JAR file and use the *javakey* program to sign it. This JAR file must be different from the one that is used with Netscape Communicator.
- For use with Microsoft Internet Explorer, you must create a CAB file that is digitally signed and specify the level of access that your code requires. You need a code-signing certificate that is compatible with the Microsoft Authenticode tools.

When using a digital certificate, the browser must recognize the Certificate Authority that signed your certificate. Either you need to obtain the certificate from a recognized authority, such as Verisign, or import your authority's certificate into the browser and mark it as trusted.

4.6.9.2 Securing the Applet Communication

The CICS Gateway for Java Version 1 classes do not use encryption when transferring data from the applet to the server. This will not be secure in an Internet environment. You may need to think of other communication options if using CICS, or wait for security features to be included with the CICS Gateway for Java. You can also use third party libraries to implement encryption of your *commareas* and database information independently of the CICS Gateway for Java or JDBC. JDBC connections do not employ encryption, and face the same problems as the CICS Gateway for Java communication.

4.6.9.3 Web Server Security

If you require authenticated access to your Web server, there are two options. You can employ basic authentication or use SSL client authentication. When the server is required to authenticate a request, you can use custom-written code to access existing security services, or design new policies using the server administration tools. For example, your server might invoke a CICS program that uses the CICS API command EXEC CICS VERIFY PASSWORD to authenticate a particular user.

4.6.10 References

The following Web sites offer usefu information already referred to in text:

- http://www.javasoft.com
- http://developer.netscape.com
- http://www.javauniverse.com/Developer/97/06/SigningApplets/index.html
- http://www.microsoft.com/java/security/jsecwp.htm
- http://www.microsoft.com/java
- http://www.microsoft.com/java/pre-sdk/contents/sdk0527.html
- http://java.sun.com/products/jdk/1.1/docs/guide/security/index.html
- http://www.verisign.com
- http://www.thawte.com
- https://certs.netscape.com
- http://www.internet.ibm.com/commercepoint/registry/index.html
- http://java.sun.com/products/jdk/preview/docs/

4.6.10.1 Related Reading

The *IBM Internet Connection Secure Server Webmasters Guide Version 2 Release 2 for OS/390* (or the other supported platforms) provides a basic introduction to Web server security. It shows how to configure the Web server for basic authentication and SSL security.

Internet Cryptography by Richard E. Smith, published by Addison-Wesley (ISBN 0-201-92480-3) is a very good source of information for anyone with very little knowledge of cryptography, but who needs to make technical decisions about cryptographic security.

The IBM Internet Web site contains lots of information on security, not only IBM security products but security in general. The page can be accessed on the Internet by the URL http://www.ibm.com/security.

4.7 JavaScript

JavaScript is a new scripting language which is being developed by Netscape. With JavaScript it is easy to create interactive Web pages. JavaScript is supported by most of todays browsers (Netscape 2.0+ and Microsoft Internet Explorer 3.0+). It is important to know that JavaScript is not the same as Java. They do have similarities but in some ways also fundamentaly differ from each other.

See the following reference for a complete description of JavaScript:

`http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/index.html`

4.7.1 JavaScript - Java Comparison

The JavaScript language resembles Java but does not have Java's static typing and strong type checking. JavaScript supports most Java expression syntax and basic control flow constructs. In contrast to Java's compile time system of classes built by declarations, JavaScript supports a runtime system based on a small number of data types representing numeric, Boolean, and string values. JavaScript has a simple, instance-based object model that still provides significant capabilities. JavaScript also supports functions without any special declarative requirements. Functions can be properties of objects, executing as loosely typed methods.

In contrast to Java, JavaScript descends in spirit from a line of smaller, dynamically typed languages like HyperTalk and dBASE. These scripting languages offer programming tools to a much wider audience because of their easier syntax, specialized built in functionality, and minimal requirements for object creation.

Table 5. JavaScript - Java Comparison

JavaScript	Java
Interpreted (not compiled) by client.	Compiled byte codes downloaded from server, executed on client.
Object-based. Uses built-in, extensible objects, but no classes or inheritance.	Object-oriented. Applets consist of object classes with inheritance.
Code integrated with, and embedded in, HTML.	Applets distinct from HTML (accessed from HTML pages).
Variable data types not declared (loose typing).	Variable data types must be declared (strong typing).
Dynamic binding. Object references checked at runtime.	Static binding. Object references must exist at compile-time.
Cannot automatically write to hard disk.	Cannot automatically write to hard disk.

4.7.2 Embedding JavaScript into HTML

JavaScript code can be embedded into HTML through 4 different methods:

- Using the SCRIPT tag.
- Specifying a file of JavaScript code.
- Using JavaScript expressions as HTML attribute values.
- Scripting event handlers.

Following we give you an overview of the first two methods. See the following reference for more information on JavaScript:

`http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/index.html`

4.7.2.1 Using the SCRIPT Tag

The `<SCRIPT>` tag is an extension to HTML that can enclose any number of JavaScript statements as shown here:

```
<SCRIPT>
    JavaScript statements...
</SCRIPT>
```

A document can have multiple SCRIPT tags, and each can enclose any number of JavaScript statements.

Because there are maybe browsers around who don't support JavaScript, the JavaScript statements can be hidden from being interpreted as text.

```
<SCRIPT>
<!-- Begin to hide script contents from old browsers.
JavaScript statements...
// End the hiding here. -->
</SCRIPT>
```

Since browsers typically ignore unknown tags, non JavaScript capable browsers will ignore the beginning and ending SCRIPT tags. All the script statements in between are enclosed in an HTML comment, so they are ignored too. JavaScript capable browsers properly interprets the SCRIPT tags and ignore the line in the script beginning with the double-slash (//).

Following is an example of an embedded JavaScript:

```
<html>
<body>
<br>
This is a standard HTML document
<br>
<script language="JavaScript">
<!-- Hide script from old browsers.

document.write("This is text from JavaScript!")

// End hiding here. -->
</script>
<p>This is text from regular HTML
</body>
</html>
```

Figure 24. Sample of an HTML embedded JavaScript

4.7.2.2 Specifying a File of JavaScript Code

The Source attribute SRC of the <SCRIPT> tag specifies a file as the JavaScript source (rather than embedding the JavaScript in the HTML). For example:

```
<HEAD>
<TITLE>My Page</TITLE>
<SCRIPT SRC="common.js">
...
</SCRIPT>
</HEAD>
```

Figure 25. Importing JavaScript code

Note that the </SCRIPT> tag is necessary!

This methode is specially usefull for sharing code/functions among different HTML pages!

JavaScript statements within a <SCRIPT> tag with an SRC attribute are ignored unless the inclusion has an error. For example, we could complete the HTML header above with an error check methode:

```
<HEAD>
<TITLE>My Page</TITLE>
<SCRIPT SRC="common.js">
document.write("Included JS file not found")
</SCRIPT>
</HEAD>
```

Figure 26. Error checking on JavaScript import function

The SRC attribute can specify any URL, relative or absolute. For example:

```
<SCRIPT SRC="http://home.netscape.com/functions/jsfuncs.js">
```

External JavaScript files cannot contain any HTML tags: they must contain only JavaScript statements and function definitions.

External JavaScript files should have the file name suffix .js, and the server must map the .js suffix to the MIME type "application/x-javascript", which the server sends back in the HTTP header.

If the server does not map the .js filename suffix to "application/x-javascript" MIME type, the browser will not load the JavaScript file specified by the SRC attribute properly. This requirement does not apply if you are using local files.

For further information on JavaScript see:

```
http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/index.html
```

or

```
http://www.webconn.com/java/javascript/intro/
```

4.8 Hypertext Markup Language

HTML is one of the major attractions of the Web. It has an architected set of tags that should be understood by all Web browsers and Web servers (although as new features are added to HTML, they may not be supported by older Web browsers). These tags are device independent. The same document can be sent to a personal computer, an AIX or UNIX machine, or a mainframe, and the Web server on each machine can understand the HTML and build the data stream to display it on the target device.

Once document writers and programmers have mastered HTML, those skills are applicable on any operating system on any machine, provided that machine has a Web browser.

Because HTML supports hypertext, it allows document writers to include links to other HTML documents. Those documents might be on the same machine as the original, or they might be on a machine on another network on the other side of the world; such is the power of HTML links.

We do not propose to document HTML itself here. For additional information, see *Using the Information Super Highway* and *Spinning the Web*. You can also find lots of information on the Web itself. One good place to start is:

http://www.w3.org/hypertext/WWW/MarkUp/MarkUp.html

4.8.1 Forms

The original specification for HTML did not provide a mechanism for the Web browser to pass information back to the Web server. Data could only flow from the Web server to the Web browser. Forms are an enhancement to HTML included in the HTML Version 2.0 draft specification. They allow data to flow from Web browser to Web server in an architected way. All new Web servers and Web browsers support forms.

Forms are analogous to CICS basic mapping support (BMS) maps (although the analogy is by no means perfect!). The Web server can prompt the user of the Web browser for information, and after the user enters that information, it is passed back to the Web server for a CGI script to process and act on the data entered into the form.

Forms are an extension of HTML and contain a set of HTML tags that allow a user to create a set of input fields in an HTML document. Each field is given a name, so that when the user enters data, the Web server can identify what data has been placed in what field. The fields are these:

- ACTION:

  Each form has an ACTION definition. This defines the CGI script on the Web server that is invoked to handle the incoming forms data when it is returned to the server. So when the Web server receives user input from the form in Figure 27 on page 68, it is the CGI script called REXXFORM that is invoked to process the request. The ACTION is resolved by the Web browser to a URL that is sent back to the Web server.

- METHOD:

  The METHOD statement on a form defines the way in which the user input for that form is passed back to the Web server There are two methods available: GET and POST.

  The results from forms specifying `METHOD=GET` are concatenated to the end of the URL string that is returned to the Web server when the user enters the data. The Web server passes the user input to the CGI script in the `QUERY_STRING` environment variable. GET is not the strategic way of passing forms data; it limits the amount of data that can be passed between the Web server and Web browser. When forms programming was first introduced it was the only mechanism available for passing from the Web browser to the Web server. For the formal HTTP specification of GET, see the following URL:

  http://www.w3.org/hypertext/WWW/Protocols/HTTP/Methods/Get.html

  `METHOD=POST` is the recommended way of passing forms data from the Web browser to the Web server. The data is not concatenated with the URL, but is sent as the body of a new document.

  The forms input data is passed to the standard input stream of the CGI script by the Web server. For the formal HTTP specification for POST see

  http://www.w3.org/hypertext/WWW/Protocols/HTTP/Methods/Post.html

4.8.1.1 Forms Programming

Forms programming is a simple and effective means of transaction processing. The flow is as follows:

1. The user enters a URL at the Web browser.
2. The request is passed to the Web server, which sends the form corresponding to the request back to the Web browser.

   Either the form is generated dynamically by a CGI script, or it can be a predefined form that is simply retrieved by the Web server. The form contains the name of the CGI script to be invoked when it is returned to the Web server.
3. The form is displayed.
4. The user enters the required data.
5. The user input is passed from the Web browser to the Web server.
6. The CGI script encoded in the form is invoked by the Web server to process the input received. If this involves sending a new form to the Web browser, we are back at Step 3.

Figure 27 on page 68 is a simple example of an HTML page containing a form. Note how easy it is to include graphics in the form using the HTML <IMG> tag, and that the form need not take up the whole of an HTML document–in fact you can have multiple forms in any document being displayed, although the Web browser can return the details of only one form to the Web server.

```
<!doctype html public "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>NAME AND ADDRESS
</TITLE> </HEAD>
<BODY>
<IMG SRC="http://ladoga.sanjose.ibm.com/slwhtml/slwimg/cwmast.gif"
     ALT="CICSWEB Masthead" ALIGN="TOP">

<FORM METHOD=POST ACTION="/cgi-bin/REXXFORM" >
<P>Enter your name and address in the fields below:
<P>Name:  <INPUT TYPE="text" NAME="name" SIZE=30 VALUE="">
<P>Address:  <INPUT TYPE="text" NAME="address" SIZE=70 VALUE="">
<P>Click here to return the form to the server:
<INPUT TYPE="submit" NAME="submit-b" VALUE="SEND">
</FORM>
</BODY></HTML>
```

Figure 27. Sample HTML Form

Figure 28 on page 69 is the CGI script that is invoked by the form in Figure 27. The script takes the name and address entered into the form, and uses them as input for a new HTML document that is built dynamically by the program itself, rather than retrieved from a file. This is an example of an "HTML-aware" application.

```
/* A Forms Program */
cl = value('CONTENT_LENGTH',,'OS2ENVIRONMENT')
data = charin(,,cl)

/* Set input fields from the incoming data stream  */
data=data'&'
do until data=''
  parse var data assign '&' data    /* Parse data for input fields        */
  parse var assign inname '=' value /* 'name=' to assign, value to value  */
  value = translate(value,' ','2b090a0d'x) /* Get rid of '+', CR, LF      */
  inname = packur(inname)           /* call packur to decode escape chars */

  /*  Put the input parameters into corresponding variables */
  select
   when inname = "name"     then name = packur(value)
   when inname = "address"  then address = packur(value)
   when inname = "submit-b" then submitb = packur(value)
   otherwise  say "<P>Unrecognised parameter passed"
  end /* select */
end

/*****************************************************************/
/*  Now use the information retrieved to build                   */
/*  a new map to be sent to the user                             */
/*****************************************************************/
say "Content-type: text/html"               /* Build HTTP Header */
say""

parse var name fname ' ' sname
say "<TITLE> Name and Address Confirmation</ETITLE>"
say "<H1>Thanks for the information"
say fname"</H1>"
say "<P> We will be sending your free sample to:"
say "<P>" address

return
```

Figure 28. REXX CGI Script to Process Form

4.8.1.2 Using Forms to Save Information about the State of Processing

The question of saving information about the state of processing is introduced in Chapter 2.5, “Matters of State” on page 11. In a transaction processing environment, the Web server sends the same form to multiple users. Because there can be multiple steps in the transaction, the server needs some mechanism to identify and track the transaction associated with the form it is currently processing.

One of the most popular techniques for saving information about the state of processing in the Web environment is to use the hidden fields facility provided by HTML forms.

Hidden fields are input fields with a type of *hidden*. They are not displayed by the Web browser and the user cannot enter data into them. They allow the Web server to store information related to the form–for example, user ID, conversation

identifier, or stage reached–in the form itself, eliminating the need to save such information on the Web server machine.

Beginning Figure 29 on page 70 to Figure 31 on page 72 shows a REXX application that uses hidden fields to save state information about the conversation taking place between the user of the Web browser and the Web server. Note the use of hidden fields to save the name of the user; in our simple case, this is the term ("handle") that uniquely identifies the user. In a more complex system, this might be a unique handle generated by the Web server itself.

The program allows the user to build a shopping list, and stores the list of items previously added to the list in the form itself in the hidden field with name `list`.

```
/*********************************************************************/
/* A Forms Program which will use information about the state of    */
/* of processing to identify the source of the request, and retrieve*/
/* any stored data associated with that user from the form.         */
/*********************************************************************/

env = 'OS2ENVIRONMENT'
cl = value('CONTENT_LENGTH',,env)

if cl > 0 then do
  newitem = ""
  list = ""
  data = charin(,,cl)

  /* Set input fields from the incoming data stream  */
  data=data'&'
  do until data=''
    parse var data assign '&' data    /* Parse data for input fields      */
    parse var assign inname '=' value /* 'name=' to assign, value to value*/
    value = translate(value,' ','2b090a0d'x) /* Get rid of '+', CR, LF    */
    inname = packur(inname)           /* call packur to decode esc chars  */

    /*  Put the input parameters into corresponding variables */
    select
      when inname = "name" then do
        name = packur(value)
        parse var name fname ' ' sname
      end
      when inname = "address"  then address = packur(value)
      when inname = "submit-b" then submitb = packur(value)
      when inname = "state"    then state = packur(value)
      when inname = "list"     then list  = packur(value)
      otherwise do
        newitem = packur(value)
        newitem = newitem||','
      end
    end /* select */
 end
```

Figure 29. (Part 1 of 3) REXX Program Using Hidden Fields

```
/*  Now determine the state of this conversation     */
/*                                                   */
/*  state=NEW means that a user has just registered. */
/*  Update the state, and send them a shopping form. */
if state="NEW" | state="OLD" then do
  if newitem \= ""  then do
    list = list||newitem
  end
  state="OLD"

  say "Content-type: text/html"
  say""
  say '<INPUT TYPE="HIDDEN" NAME="state" VALUE='state'>'
  say'<HTML><HEAD><TITLE>Display Shopping List'
  say'</TITLE> </HEAD>'
  say'<IMG SRC="/icons/cwmast.gif"'
  say'ALT="CICSWEB Masthead" ALIGN="TOP">'
  say'<FORM METHOD=POST ACTION="/cgi-bin/rcgi3" >'
  say '<INPUT TYPE="HIDDEN" NAME="state" VALUE="'state'">'
  say '<INPUT TYPE="HIDDEN" NAME="name" VALUE="'name'">'
  say '<INPUT TYPE="HIDDEN" NAME="list" VALUE="'list'">'
  say '<INPUT TYPE="HIDDEN"  NAME="address" VALUE="'address'">'

  len = length(list)
  if len > 0 then do
  /* IF there is already a list for this user, display the contents */
    say'<P>Here are the items currently recorded on your shopping list:'
    say'<UL>'
    data = list
    do until data=''
      parse var data item ',' data
      if item \='+' then
        say'<LI>'translate(item)
    end /* data loop */
    say'</UL>'
  end /* display list */
  else  /* No list currently exists for this user */
    say'<P>There are no items currently on your shopping list'

  say'<P>Enter items you wish to add to the list here, separating'
  say'them with a comma(,):'
  say'<P>Name:  <INPUT TYPE="text" NAME="item" SIZE=30 VALUE="">'
  say'<P>Click here to add this input to the list:'
  say'<INPUT TYPE="submit" NAME="submit-b" VALUE="SEND">'
  say'</FORM>'
 end
end
```

Figure 30. (Part 2 of 3) REXX Program Using Hidden Fields

```
  else
  /*********************************************************/
  /*  There is no forms data to be read in, so this must be */
  /*  a new conversation. Ask the user to enter the name    */
  /*  and address so that we can retrieve any data held     */
  /*********************************************************/
  do
    say "Content-type: text/html"
    say""

    say'<!doctype html public "-//IETF//DTD HTML 2.0//EN">'
    say'<HTML><HEAD>'
    say'<TITLE>NAME AND ADDRESS'
    say'</TITLE> </HEAD>'
    say'<IMG SRC="/icons/cwmast.gif"'
    say'ALT="CICSWEB Masthead" ALIGN="TOP">'

    say'<FORM METHOD=POST ACTION="/cgi-bin/rcgi3" >'
    say '<INPUT TYPE="HIDDEN" NAME="state" VALUE="NEW">'
    say'<P>Enter your name and address in the fields below:'
    say'<P>Name:  <INPUT TYPE="text" NAME="name" SIZE=30 VALUE="">'
    say'<P>Address:  <INPUT TYPE="text" NAME="address" SIZE=70 VALUE="">'
    say'<P>Click here to return the form to the server:'
    say'<INPUT TYPE="submit" NAME="submit-b" VALUE="SEND">'
    say'</FORM>'
  end /* No forms input on call */

 return
```

Figure 31. (Part 3 of 3) REXX Program Using Hidden Fields

Saving information about the state of processing in forms does, however, have some drawbacks:

- All the information about the state of processing in the form has to be sent across the network to the Web browser and back again. If there is a large amount of information about the state of processing (for example, our shopping list might be very big), this will generate a lot of extra traffic across the network.

  This also has security implications, because the information about the state of processing might contain sensitive data that you do not want to flow across the network.

- The information about the state of processing will survive for only as long as the current dialog between Web server and Web browser lasts. If for some reason the user of the Web browser decides to close down and restart the Web browser, all the information about the state of processing in the form is effectively lost.

You may therefore decide that your information about the state of processing is too important to reside solely in the form itself, and keep all or some of it on the Web server. Beginning with Figure 32 on page 73 to Figure 34 on page 75 illustrates how this might be done in an OS/2 environment using a CGI script. The information about the state of processing (in this case our shopping list) is saved in a file on the Web server machine, and the only information about the state of

processing required in the form is a unique identifier that is used to identify the stored information about the state of processing for this conversation.

```
 /*******************************************************/
/* A Forms Program that will use information about the   */
/* state of processing to identify the source of the     */
/* request, and retrieve any stored data associated with */
/* that user from the form.                              */
/*******************************************************/
/*      If there is any forms data for us to read    */
/*      read it in and pass it.                      */
env = 'OS2ENVIRONMENT'
cl = value('CONTENT_LENGTH',,env)

if cl > 0 then do
  newitem = ""
  data = charin(,,cl)

  /* Set input fields from the incoming data stream  */
  data=data'&'
  do until data=''
    parse var data assign '&' data    /* Parse data for input fields      */
    parse var assign inname '=' value /* 'name=' to assign, value to value*/
    value = translate(value,' ','2b090a0d'x) /* Get rid of '+', CR, LF    */
    inname = packur(inname)           /* call packur to decode esc chars  */

    /*  Put the input parameters into corresponding variables */
    select
     when inname = "name" then do
       name = packur(value)
       parse var name fname ' ' sname
     end
     when inname = "address" then   address = packur(value)
     when inname = "submit-b" then  submitb = packur(value)
     when inname = "state" then     state = packur(value)
     otherwise do
       newitem = packur(value)
       newitem = newitem||','
     end
    end /* select */
 end
```

Figure 32. (Part 1 of 3) CGI Script Saving Data on the Web Server

```
  /*  Now determine the state of this conversation     */
  /*  state=NEW means that a user has just registered. */
  /*  Update the state, and send them a shopping form. */
  if state="NEW" | state="OLD" then do
    if newitem \= ""  then do
      result = charout(fname)
      result = charout(fname,newitem)
      result = charout(fname)
    end
    state="OLD"

    say "Content-type: text/html"
    say""

    say '<INPUT TYPE="HIDDEN" NAME="state" VALUE='state'>'
    say'<HTML><HEAD><TITLE>Display Shopping List'
    say'</TITLE> </HEAD>'
    say'<IMG SRC="/icons/cwmast.gif"'
    say'ALT="CICSWEB Masthead" ALIGN="TOP">'
    say'<FORM METHOD=POST ACTION="/cgi-bin/rcgi4" >'
    say '<INPUT TYPE="HIDDEN" NAME="state" VALUE='state'>'
    say '<INPUT TYPE="HIDDEN" NAME="name" VALUE='name'>'
    say '<INPUT TYPE="HIDDEN"  NAME="address" VALUE='address'>'

    result = charin(fname,1,0)
    len = chars(fname)

    if len > 0 then do
    /* IF there is already a list for this user, display the contents */
      say'<P>Here are the items currently recorded on your shopping list:'
      say'<UL>'
      data = charin(fname,1,len)
      data = data','
      do until data=''
        parse var data item ',' data
        if item \='+' then
          say'<LI>'translate(item)
      end /* data loop */
      say'</UL>'
    end /* display list */
    else           /* No list currently exists for this user */
      say'<P>There are no items currently on your shopping list'
    say'<P>Enter items you wish to add to the list here, separating'
    say'them with a comma(,):'
    say'<P>Name:  <INPUT TYPE="text" NAME="item" SIZE=30 VALUE="">'
    say'<P>Click here to add this input to the list:'
    say'<INPUT TYPE="submit" NAME="submit-b" VALUE="SEND">'
    say'</FORM>'
   end
end
```

Figure 33. (Part 2 of 3) CGI Script Saving Data on the Web Server

```
 else
 /*********************************************************/
 /*  There is no forms data to be read in, so this must be */
 /*  a new conversation. Ask the user to enter the name    */
 /*  and address so that we can retrieve any data held.    */
 /*********************************************************/
 do
   say "Content-type: text/html"
   say""

   say'<!doctype html public "-//IETF//DTD HTML 2.0//EN">'
   say'<HTML><HEAD><TITLE>NAME AND ADDRESS'
   say'</TITLE> </HEAD>'

   say'<IMG SRC="/icons/cwmast.gif"'
   say'ALT="CICSWEB Masthead" ALIGN="TOP">'

   say'<FORM METHOD=POST ACTION="/cgi-bin/rcgi4" >'
   say '<INPUT TYPE="HIDDEN" NAME="state" VALUE="NEW">'
   say'<P>Enter your name and address in the fields below:'
   say'<P>Name:  <INPUT TYPE="text" NAME="name" SIZE=30 VALUE="">'
   say'<P>Address:  <INPUT TYPE="text" NAME="address" SIZE=70 VALUE="">'
   say'<P>Click here to return the form to the server:'
   say'<INPUT TYPE="submit" NAME="submit-b" VALUE="SEND">'
   say'</FORM>'
 end /* No forms input on call */

return
```

Figure 34. (Part 3 of 3) CGI Script Saving Data on the Web Server

4.9 Utility CGIUTILS

For those using a CERN-based Web server such as the Domino Go Webserver, the `cgiutils` utility provides an easy means of building HTTP headers to be sent in your HTTP response to the Web browser request. Figure 35 shows how you might call `cgiutils` to create an HTTP header specifying the content type for an HTML document.

`Utility cgiutils` simply builds the appropriate header and writes it to standard output. If you are familiar with the HTTP header format, you may prefer to build the headers inside your own application. Doing this in your application is likely to provide better performance, because it avoids the overhead of invoking a separate program. Note that header information must be written to the standard output device before you start writing out the HTML text itself.

```
/* Use cgiutils to build HTTP header */
'@cgiutils -ct text/html'
say ''
say "<H1>Hello World</H1>"
say "<P>Is there anybody out there ?"
return
```

Figure 35. Example of REXX CGI Script Using CGIUTILS

4.10 Utility CGIPARSE

The CERN-based Web servers provide another utility called `cgiparse`. This utility helps you to parse incoming forms data. Outputs from `cgiparse` is written to the standard output device; so you need to pipe its output into your own program so that you can make use of its output. We found it easier to parse the data in the application itself (see Figure 28 on page 69 for a simple example of parsing input variables from a form). Doing it in your application is likely to provide better performance, because it avoids the overhead of invoking a separate program.

Figure 36 shows how you can use `cgiparse` to extract form data into REXX variables in your CGI script.

```
/* call cgiparse to get form input data */
say "Content-type: text/html"                    /* Build HTTP Header */
say ""

"@cgiparse -form -prefix VAR. | rxqueue"       /* Parse form variables */
parse pull result /* Format is VAR.var1='val 1'; VAR.var2='val 2'; etc.*/
interpret result  /* result string is valid REXX syntax, so just do it */
say '<pre>Result="'result'"'
say 'VAR.submit     ="'var.submit'"'
say 'VAR.input_data ="'var.input_data'"</pre>'
return ''
```

Figure 36. Example of REXX CGI Script Using CGIPARSE

4.11 Caching

When programming for the Web, you need to be aware that to minimize the flows across the network, proxy servers usually cache frequently requested documents rather than retrieving them every time they are requested. You need to bear this in mind when coding your Web application. If you have an application for which this can cause problems, you can use the `pragma` HTTP header when sending your response (see 4.2, “Hypertext Transfer Protocol Header Information” on page 35) to specify the `no-cache` parameter. This tells any proxy server between your Web server application and the Web browser not to cache this document, but always get a fresh copy.

When processing a Web browser request, the `If-Modified-Since` HTTP header, which can be used for forms using GET, is also useful, because it tells the Web server that it needs to return the document only if it has been modified since the specified date.

Most Web browsers also use caching. If you request a document that was retrieved earlier in your Web browser session, the Web browser redisplays the earlier document. This can lead to confusion if the document is also a form. You need to ensure that your Web forms applications can cope with being called with spurious forms at unexpected times.

Chapter 5. Accessing CICS/ESA from the Web

A CICS application usually consists of different programs, queues, files and databases. The user of a CICS application typically works on a 3270 screen or with an ECI-connected program. Each user interaction starts CICS application programs within one CICS execution unit and may access/update data from files, databases, or other shared objects that can be accessed within a transaction. The CICS Transaction Manager has to keep track of the state of processing.

As the demand for Internet and Intranet services grows, more and more companies are considering making their existing CICS/ESA applications fully or partially available to the Web.

Although the known Web programming techniques can be used when accessing CICS/ESA, the dividing function between Web client, Web server, and CICS raises new application design issues:

- Which component is responsible for maintaining information about the state of processing?
- Which component is responsible for security?
- Which component builds and interprets the HTML documents, and analyses user input?
- Which component makes the decision to commit changes to CICS-owned resources?
- Which component is responsible for restarting the conversation in the event of a failure?

As you design your CICS/ESA Web application, you must decide how to split responsibilities among the components. You can delegate as much responsibility as possible to the CGI script, the CICS Gateway for Java, or even to the Web client, and only invoke CICS/ESA to perform an update to a CICS resource. Alternatively, you can do everything in the CICS/ESA environment, and only use the CGI script or CICS Gateway for Java or the CICS Web Interface to pipe HTML input and output between the Web browser and CICS/ESA.

Given that CGI scripts and CICS Gateway for Java applets are more suited to some tasks than CICS/ESA, and CICS/ESA is better suited to others, the best solution for your system is likely to be one that allocates to each component the functions it performs best.

The way you split responsibilities between your Web server and CICS/ESA will in part be dictated by how the two are connected. For example, advanced program-to-program communication (APPC) and TCP/IP connections between a Web server and CICS/ESA implement security in different ways. We therefore look at some of the ways you can connect your Web server to CICS/ESA.

We then look at some of the different types of CICS/ESA applications that can be invoked from the Web, and illustrate some programming techniques we found useful when accessing CICS/ESA applications from the Web.

5.1 Connecting CICS to the Internet

The nature of the interface between CICS/ESA and the Web is to some extent dictated by the infrastructure connecting the two. There are different ways to connect a Web server to CICS/ESA. We identify and discuss the following approaches to access CICS applications from the Web:

- **Web Server programming**
  - CICS Internet Gateway for 3270 applications
  - OS/390 Web Server with a CGI sample for CICS
- **Client-side programming with CICS Gateway for Java**
  The Java gateway provides two-tier access from a Java program to a CICS application. Any Java-enabled browser or Network Computer can dynamically download a Java applet and access CICS data transparently.
  All of the well-known Java benefits are available. The code is downloaded to the client from the server when it is needed, so there are no problems of version control. Java allows a full range of GUI techniques, and applications can be run unchanged on almost any client, completely independent of the server.
- **Integrating the HTTP protocol with the CICS Web Interface**
  The CICS Web Interface opens the OS390 or the MVS/ESA system for direct Web access

These approaches can either be used through direct connections from a Web browser to the CICS/ESA applications or utilizing indirect connections through a gateway represented by other members of the CICS family of products.

Table 6 summarizes the alternative offerings for connecting CICS to the Internet.

Table 6. Offerings for Connecting CICS to the Internet

Offering	Use	Availability
EXCI CGI Sample Program	Well suited to general Internet access to CICS/ESA	SupportPac can be downloaded without charge from WWW
CICS Internet Gateway	Supports any 3270 application	Included in: IBM Transaction Servers IBM Internet Connection Servers IBM CICS Clients V2 IBM Connectors
CICS Gateway for Java	Supports any Java-enabled Web browser	Download without charge from WWW. Included in: IBM CICS Clients V2 IBM Connectors

Offering	Use	Availability
CICS Web Interface	Provides direct connection to CICS/ESA. Well suited to Intranet applications. The CWI may be used together with the 3270 Bridge to run unmodified terminal-oriented transactions.	Included in IBM CICS Transaction Server for OS/390. Installable function of CICS/ESA V4.1 from 11/96

5.1.1 EXCI CGI Sample Program

The EXCI CGI sample program in Figure 37 provides an interface between the IBM Internet Connection Server for MVS/ESA (running on MVS Open Edition) and CICS/ESA. This sample may be tailored by the user to meet specific application needs.

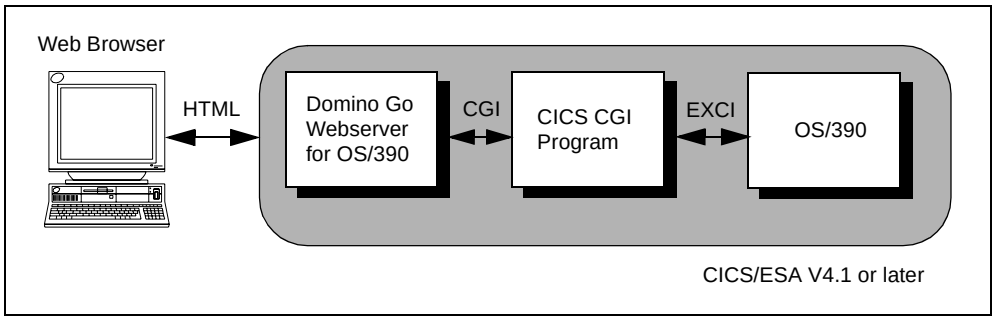


Figure 37. EXCI CGI Sample Program

5.1.2 CICS Internet Gateway

An IBM-provided CGI script that makes a Web browser appear like a 3270 terminal and hence provides Internet access to existing 3270 applications is shown in Figure 38. Works in conjunction with the IBM CICS Client.

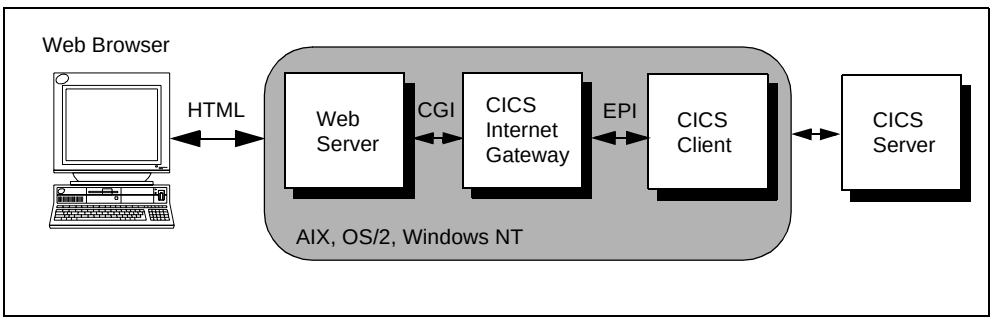


Figure 38. CICS Internet Gateway

5.1.3 CICS Gateway for Java

The CICS Gateway for Java, shown in Figure 44, is a Java application that allows a Java-enabled Web browser or network computer to download a Java applet and transparently access CICS data and applications. It works in conjunction with the IBM CICS Client.

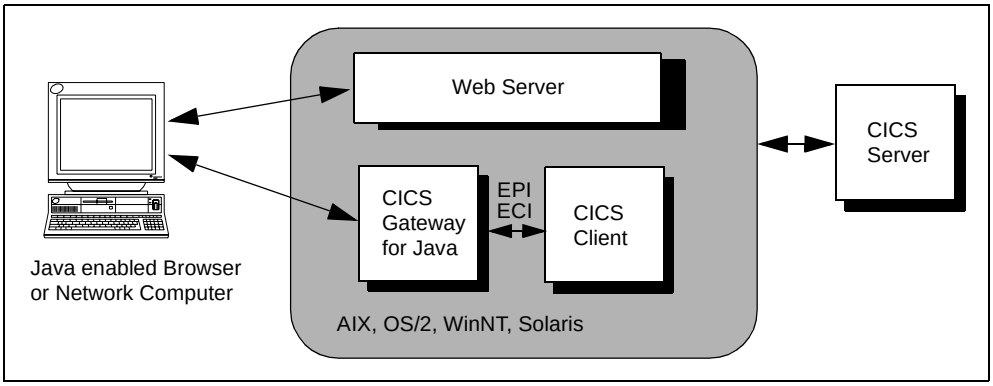


Figure 39. CICS Gateway for Java

5.1.4 CICS Web Interface

The CICS Web interface, shown in Figure 40, offers support in the CICS Transaction Server for OS/390 and provides a TCP/IP connection directly into CICS, without going through a Web server or gateway. This is the most efficient way for a Web browser to access CICS/ESA and requires no further products other than TCP/IP for MVS.

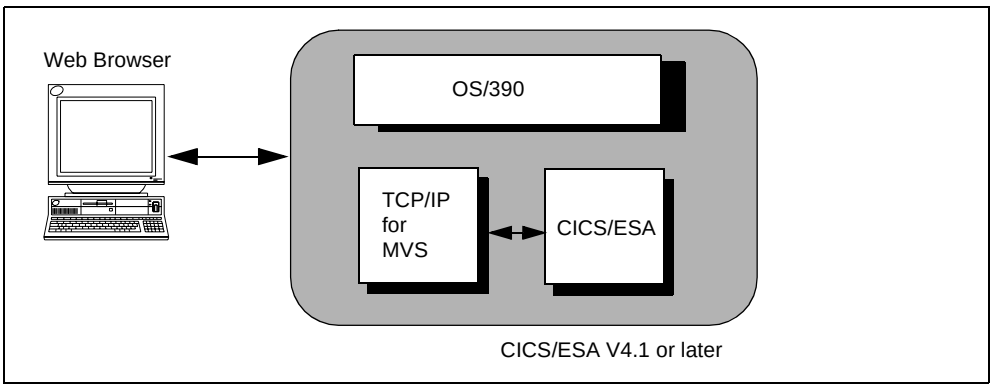


Figure 40. CICS Web Interface

5.2 CICS Access Overview

Figure 41 on page 81 gives you a general overview on how to access CICS transaction processing services from outside CICS, particularly from the Web. In the following subsections, we concentrate to the most common Web access methods only, knowing that many other access ways may be constructed. At the end of this chapter we suggest what to consider while constructing special CICS access paths.

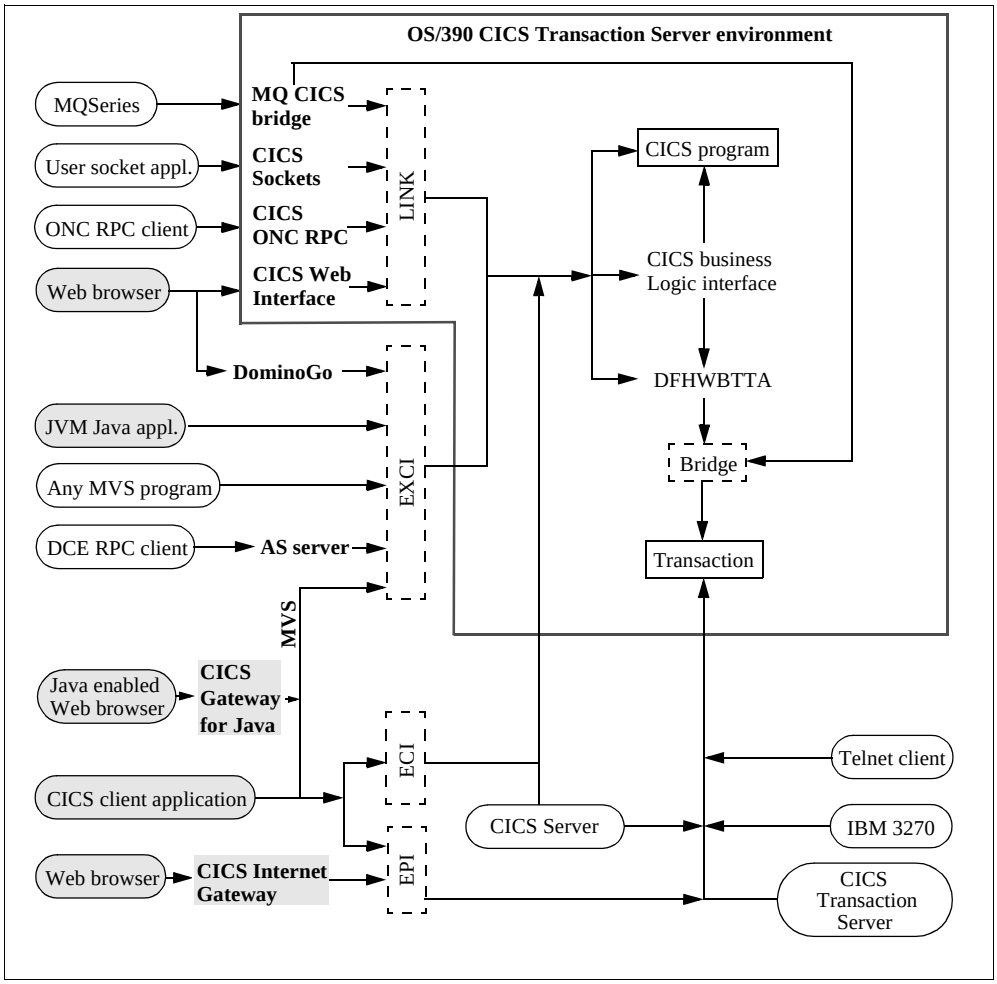


Figure 41. Map showing How to Get CICS Transaction Service

In Figure 41 on page 81 the rounded boxes show the possible external requests. The plain boxes are targeted by the external requests and represent CICS programs and CICS transactions. The broken lines enclose CICS provided interfaces.

The shaded boxes represent the most common methods of accessing CICS from the Web. Future sections that follow will focus on these methods.

5.2.1 CICS/ESA direct Web Connection Solutions

Accessing CICS/ESA applications through either Internet or Intranet, may bring up the wish of direct access from a Web browser to CICS/ESA. This lowers the distributed infrastructure and its related maintenance because there is no gateway or intermediate Web server needed. However, using these facilities assumes a clear understanding of all possible security issues to a project such as any existing security guidelines, or any possible new security threats while opening an internal production systems to the Internet or Intranet.

5.2.1.1 The CICS Web Interface

The CICS Web Interface is a two-tier method for connecting Web browsers to CICS Transaction Server. It supports the separation of presentation logic, utilizing

any Web browser, from business logic in application design. CWI is available as an integral part of CICS for MVS/ESA V4.1 and CICS Transaction Server for OS/390. It can be used for non-terminal CICS transactions as well as starting a CICS terminal-oriented transaction. In this case, CICS provides a procedure and supporting tools for mapping 3270 data streams into and out of HTML. This direct access technique performs very well, which makes it especially useful on Intranet projects where existing 3270-based applications have to be brought unmodified to a Web browser. The CWI may be used with the Domino Go Webserver to provide support for the secure Web protocol SSL, allowing encryption of data across the network.

In Figure 42 on page 82 the Web browser constructs an HTTP request which is passed across the network to TCP/IP for MVS in the server. TCP/IP for MVS relays the request to the Domino Go Webserver, which uses CICS provided code and user-provided definitions to construct a request for the CICS Web Interface and route it to the CICS service. If ICSS is bypassed, the CWI links to the different services needed to prepare and start the CICS request.

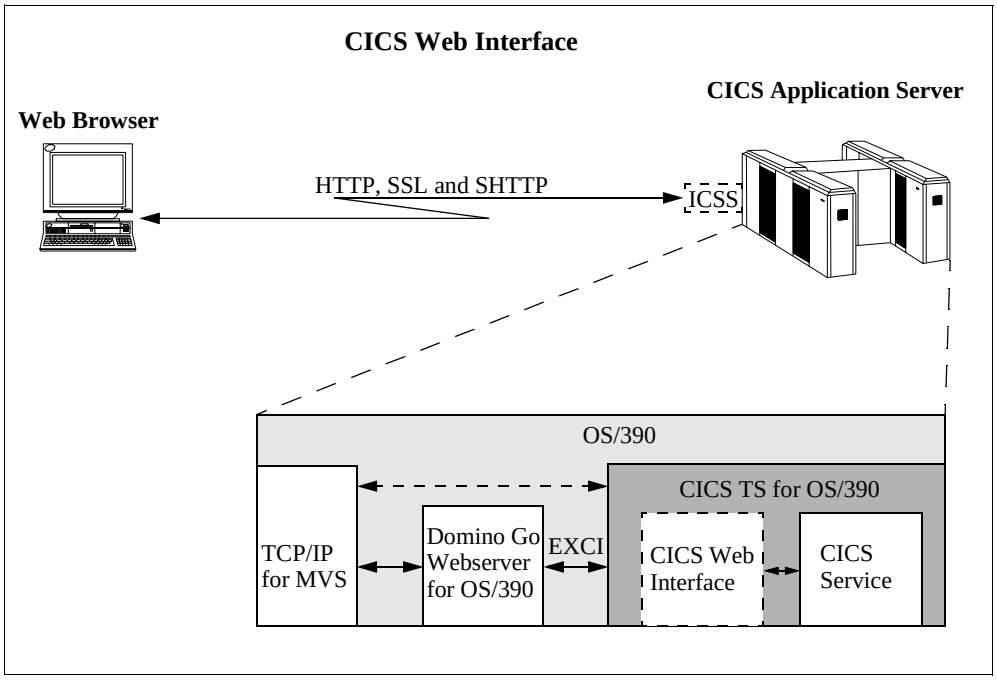


Figure 42. Accessing CICS/ESA through the CWI

5.2.1.2 CICS Web Interface Control Flow

Figure 43 on page 83 shows the control flow through the CICS Web interface to a CICS Transaction without using Domino Go Webserver.

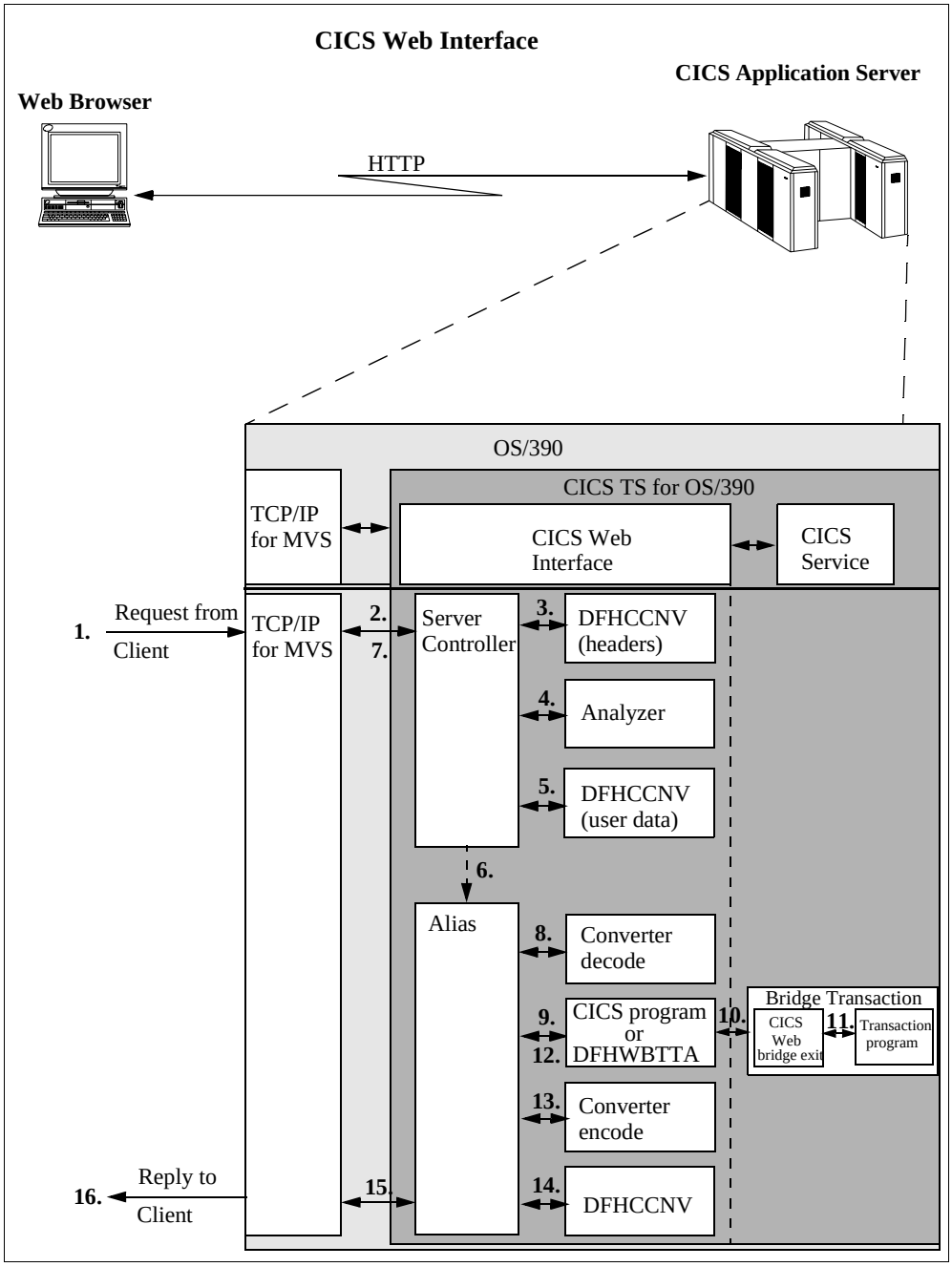


Figure 43. Control Flow of a CWI Transaction

The steps to process a CWI transaction are these:

1. An HTTP request arrives in TCP/IP for MVS from a Web browser.
2. The server controller monitors the TCP/IP for MVS interface for incoming HTTP requests, and TCP/IP passes the request to it.
3. The server controller calls DFHCCNV to translate the HTTP request headers from ASCII to EBCDIC. (HTTP headers are always transmitted in ASCII.)
4. The server controller links to the analyzer. The analyzer interprets the request and specifies the name of the alias transaction and the CICS program that is to service the request.

5. If the analyzer requests data conversion, the server controller calls DFHCCNV to translate the user data in the HTTP request from the client code page to EBCDIC. (HTTP user data is transmitted in the client code page.)
6. The server controller starts an alias transaction to deal with all further processing of the request in CICS.
7. The server controller returns to monitor the TCP/IP for MVS interface for HTTP requests.
8. If the analyzer requests a converter, the alias calls it, requesting the Decode function. Decode can modify the communication area for the CICS request. If a terminal-oriented transaction is called, Decode sets up the communication area for DFHWBTTA.
9. The alias calls DFHWBTTA. In case of a non-terminal-oriented transaction, at this point the CICS program would be called. The communication area passed DFHWBTTA is the one set up by Decode. If no converter program was called, the communication area contains the entire request.
10. DFHWBTTA finds the transaction ID for the terminal oriented transaction and starts the CICS-supplied bridge transaction that runs the CICS Web bridge exit, DFHWBLT.
11. The CICS Web bridge exit passes the target transaction ID to the CICS-supplied transaction, and the corresponding transaction program is run.
12. The output is returned to the communication area.
13. When the program attempts to write to its principal facility, the data is intercepted by the CICS Web bridge exit, and returned to the alias. If the caller requested a converter, the alias calls the Encode function of the converter, which uses the communication area to prepare the response. If no converter program was called, the alias assumes that the communication area contains the desired response.
14. If the analyzer requested data conversion, the alias calls DFHCCNV to translate the HTTP response.
15. The alias returns the results to TCP/IP for MVS and ends.
16. The response is sent back to the Web browser.

5.2.1.3 CWI Security

The CICS security facilities and the Domino Go Webserver security facilities can be used for user authorization and access control between the browser and the requested transactions. However, unless we do use the CWI through the Domino Go Webserver, every connection from a browser to CWI must be considered as unsecure since the data stream (including user ID and password) is encoded only. Secure connections through SSL, where all data sent are being encrypted, can be achieved using CWI through Domino Go Webserver.

5.2.2 The CICS Gateway for Java

The CICS Gateway for Java is a Java application which resides on a Web server with a CICS Client. Complementing the Gateway are CICS Java classes used in applets that execute in browsers or network computers.

The CICS Gateway for Java provides three Java classes:

1. Gateway connections to CICS systems through the JavaGateway class.

2. External Call Interface (ECI) to CICS through the ECIRequest class.
3. External Presentation Interface (EPI) to CICS through the EPIRequest class.

Although the EPIRequest class can be downloaded from CICS Gateway for Java on MVS, only the ECIRequest class is supported connecting for a Java client to the CICS Gateway for Java on MVS.

5.2.2.1 Operation of the CICS Gateway for Java

The CICS Gateway for Java, Figure 44, is a multithreaded TCP/IP server application that runs separately from CICS. The operation of the gateway can be shown best by walking through the process step by step:

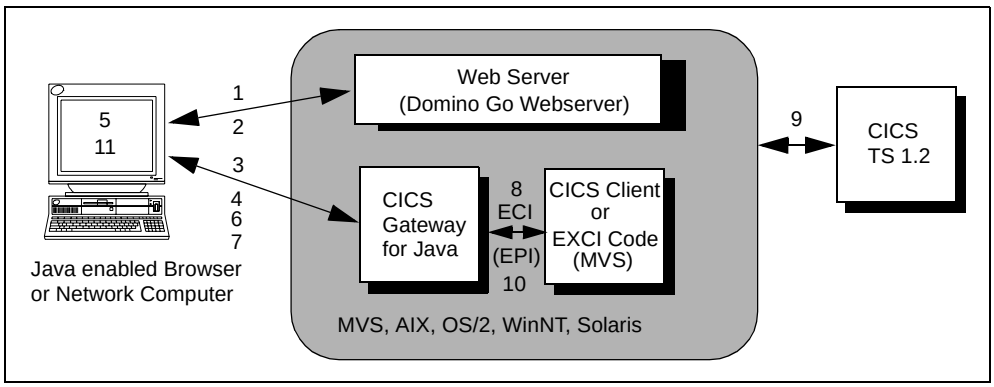


Figure 44. CICS Gateway for Java

1. The Web browser or the network computer requests an HTML page from a Web server using the HTTP protocol, which contains a Java applet tag.
2. The Web server returns the requested HTML page containing the tag that identifies a Java applet.
3. The browser now starts loading the applet and requesting related Java classes from the Web server.
4. The Web server returns the requested Java classes including the CICS Gateway for Java classes.
5. When all classes are returned, the browser starts the applet
6. The Java applet on the browser now creates a JavaGateway object to connect to the CICS Gateway for Java. This establishes communication between the browser and the long-running Gateway process using Java's socket protocol. Furthermore, at this time an LUW token is being created.
7. The Java applet creates an ECIRequest (or EPIRequest) object containing ECI (or EPI) calls and sends it to the Gateway using the JavaGateway flow method. The ECIRequest (or EPIRequest) supports most of the ECI (or EPI) call parameters.
   If the CICS Gateway for Java runs on OS390, only the ECIRequest object is supported.
8. The CICS Gateway for Java now receives the request, unpacks it, and makes a corresponding ECI (or EPI) call to the CICS Client, which forwards the call to the intended CICS server.
   When the CICS Gateway for Java is running on OS/390, ECI and EPI interfaces are not available. The CICS client is not available on OS/390 and

CICS Transaction Server does not provide these interfaces to OS/390 programs. In this case, CICS Gateway for Java on OS/390 actually uses the External CICS Interface (EXCI), which is provided by the CICS Transaction Server. The applets that are clients to the CICS Gateway for Java on OS/390 still flow ECIRequest data, but it is turned into EXCI calls to CICS TS. This allows the applet code to remain unchanged regardless of where the CICS Gateway for Java resides and which CICS server is used.

9. The CICS server processes the call, including verifying the user ID and password if required, and passes control and user data to the application program.
10. When through processing, the application returns control and data back to CICS which returns the requested data to the Gateway through the CICS Client.
11. The Gateway packs these results and returns them to the Java applet running on the Web browser.

Figure 45 on page 86 shows how the CICS Gateway for Java can run on a workstation and use the CICS Client to access CICS servers in a three-tier configuration.

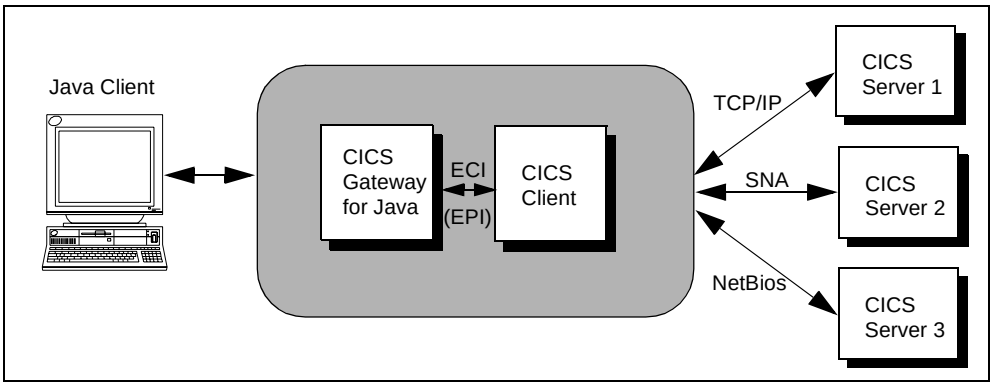


Figure 45. External CICS Gateway for Java

Figure 46 on page 86 shows the CICS Gateway for Java running on the same workstation as the CICS server, in a two-tier configuration, without the need for a CICS Client.

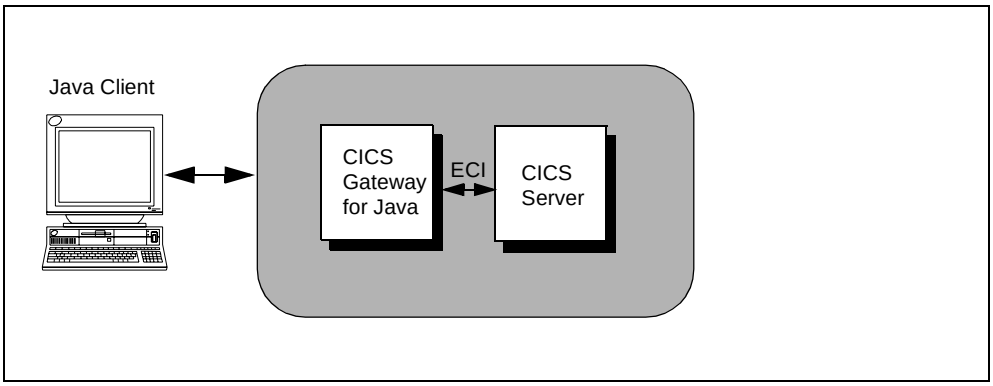


Figure 46. CICS Gateway for Java on the Same System with CICS Server

Figure 47 on page 87 shows the CICS Gateway for Java running on OS/390 accessing CICS TS using the EXCI. In Figure 45 on page 86, the CICS Client can access multiple servers simultaniously using different protocols. In Figure 46 on page 86, the CICS Gateway for Java can access only the single local server, and no other CICS servers. In Figure 47 on page 87 the use of the EXCI allows the CICS Gateway for Java to access multiple CICS TS regions, but not other CICS workstation servers.

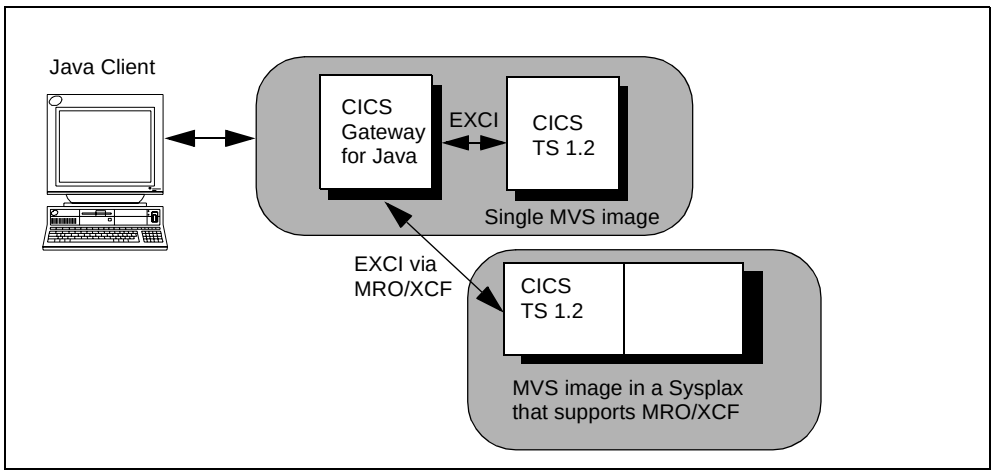


Figure 47. CICS Gateway for Java on OS/390

5.2.2.2 The CICS Gateway for Java Programming Interface

The CICS Gateway for Java provides five classes through which you can access CICS programs from Java applets or applications:

- ibm.cics.jgate.client.JavaGateway
- ibm.cics.jgate.client.ECIRequest
- ibm.cics.jgate.client.EPIRequest
- ibm.cics.jgate.client.Callbackable
- ibm.cics.jgate.client.GatewayRequest

When an instance of the JavaGateway class is created, the constructor method requires the TCP/IP address and port number of the CICS Gateway for Java to which a connection will be made. The object opens a TCP/IP socket connection to the specified CICS Gateway for Java and this remains open until the close method is called, or the object is destroyed. The GatewayRequest class is superclass of ECIRequest and EPIRequest and contains variables common to both. The Java program creates instances of ECIRequest and EPIRequest and these objects are used as parameters to the flow method of the JavaGateway object. The JavaGateway class supports two kind of connections:

1. A network gateway connection, which is a Java client-only connection to a CICS Gateway for Java (remember that CICS Gateway for Java on MVS supports only ECIRequest class for transactions).
2. A local gateway connection which, despite its name, does not need a CICS Gateway for Java at all. Instead, the local gateway connection runs only on systems that have a CICS client communicating to the CICS system or a CICS pseudo-client on MVS using EXCI to communicate with the CICS system.

Again, if the Java application runs on MVS using the CICS pseudo-client, only the Java ECIRequest class is supported.

The particular Logical Unit of Work (LUW) token is created and returned by CICS on the first CICS Gateway for Java invocation. The client application must supply the LUW token on all subsequent requests to CICS within the same LUW.

The EPIRequest object represents calls to the ECI. The public variable Call_Type defines which EPI call has to be done. The results of the EPI call are returned in the object after the use of the flow method with the JavaGateway object.

The Callbackable class is a Java interface that is used when you require an asynchronous call with callback. You need to write a class that implements the Callbackable interface and pass an instance of this class to the setCallback method of the request object. When flow is called it returns immediately and when the data is actually returned by the CICS Gateway for Java, the Callbackable object is run in a new thread.

5.2.2.3 Browser Session

The browser session to the CICS Gateway for Java is initiated with the first ECI request from a CICS Gateway for Java for a LUW. During the browser session with the CICS Gateway for Java, the Gateway maintains all state information in the form of variables held in memory. The LUW token is transmitted between the CICS client component and the CICS Gateway for Java as part of the explicit ECI parameters.

A browser session with the CICS Gateway for Java is terminated and the corresponding LUW is committed when the Gateway:

- Responds to a client‘s commit request.
- Responds to a client‘s rollback request.
- Detects a timeout and sends a rollback request to the CICS program.

5.2.2.4 Security

The CICS facilities for user authorization and access control between the client for Java and the CICS Gateway for Java can be used. Although using SSL security on the Web server enables Java code to be downloaded securely, this mechanism does not automatically provide security for any communication that the applet code then independently performs. The IBM CICS Gateway for Java Version 1 classes do not yet use encryption when transfering data from the applet to the server. This will not be secure in an Internet environment! You may need to think of other communication options if using CICS, or wait for security features to be included with the CICS Gateway for Java.

A current design consideration in the use of Java applet communication on the public Internet is the impact of firewalls. *Firewalls* is the term given to a configuration of software which prevents unauthorized traffic between a trusted network and an untrusted network. We discuss the technical aspects of firewalls in Chapter 5.5.3.1, “Placing the Firewall” on page 104. Firewalls are put in place to protect company assets from outside intrusion, but they can also limit legitimate communications. Firewalls come into play in two ways: the general accessibility of a server to outside users-(inbound restrictions), and the ability of end users inside a firewall to perform certain network functions outside their firewall-(outbound restrictions).

The CICS Gateway for Java configuration is well suited to avoid problems from outside intrusion since the gateway processor can be placed outside the firewall and be connected through the firewall to the CICS server. Outbound firewalls, however, can be a problem. Large companies may use a firewall to limit the types of connections or protocols that may be used and this can affect the operation of the Gateway.

In summary, the use of Java on an Intranet works very well since firewalls are typically not a factor. However, in designing Internet applications for end users outside a company, consider whether those firewalls will be an inhibitor. If so, alternative processing for those users, such as executing the Java code as a Java application on the Web server (sometimes known as a *servlet*), could be considered. As with other current limitations in using the Internet, firewall technology can be expected to evolve rapidly over the next few years to address this design consideration.

5.2.3 The CICS Internet Gateway

The CICS Internet Gateway allows you to use a Web browser to access CICS/ESA 3270 applications.

The CICS Internet Gateway is a CGI script that takes HTML input from the Web browser, maps it to the 3270 data stream, and uses the EPI to send it to a CICS server from where it is routed to CICS/ESA. It then intercepts the 3270 data stream returned by the CICS/ESA application, looking for variable data fields, static text, and cursor position. This data stream is converted to an HTML document and sent to the Web browser. The process not only enables any Web browser to act as a 3270 terminal but can also add a more appealing GUI to 3270 applications.

Most existing CICS 3270 applications should be able to use the CICS Internet Gateway without modification.

You can add value to your existing 3270 applications by providing header and trailer information and help information in HTML format that can be merged with the mapped 3270 data. Although HTML does not directly support 3270 features such as PF keys, attention identifier keys, and cursor position, nor the GUI concept of action bars, the CICS Internet Gateway has features to help you to simulate these functions.

Even if you do not intend to allow Internet users access to your CICS/ESA system, the CICS Internet Gateway can still be very useful as a means of providing an improved user interface, using the Web browser as an alternative to a more conventional 3270 emulator. No CICS-specific software is needed on the Web browser system.

5.2.4 Other Ways to Access CICS/ESA

While the CICS family of products provides an integrated solution to accessing CICS/ESA from a workstation environment, you can also access CICS/ESA using a number of other mechanisms.

5.2.4.1 CICS/ESA and TCP/IP for MVS Sockets Interface

If you are familiar with TCP/IP sockets programming, you can issue socket calls from your CGI script to communicate directly with a CICS/ESA application. If you

are using GoServe, you can use the RXSOCK package available through IBM Employee Written Software to issue socket calls directly from your REXX filter.

See Chapter 10, “CICS Sockets Sample” on page 153 for an example of an application that uses the TCP/IP for MVS sockets interface.

5.2.4.2 APPC Programming

You can issue APPC calls from your CGI script or GoServe filter to communicate directly with a CICS/ESA distributed transaction processing (DTP) transaction. This is a more secure environment than sockets, since you can use the security features of APPC.

Once in CICS/ESA, your DTP application can, if required, issue FEPI calls to connect to a 3270 application.

5.2.4.3 Remote Procedure Calls

If you are using a DCE environment, you can use DCE remote procedure calls (RPCs) to route requests from your Web server to CICS/ESA. DCE uses the CICS/ESA EXCI to perform a distributed program link to a CICS/ESA program. DCE provides a more secure environment than native sockets, as discussed in 3.3.7, “Kerberos” on page 25.

IBM has also announced the Open Network Computing RPC (ONC RPC) feature for CICS/ESA, which implements the RPC model developed by Sun Microsystems. This feature allows ONC RPC clients to invoke CICS/ESA applications directly. You could therefore code ONC RPC calls in your CGI script to call your CICS/ESA server application. The ONC RPC interface is included in most TCP/IP implementations. Figure 48 shows how you can connect a Web server to CICS/ESA using the CICS/ESA ONC RPC feature.

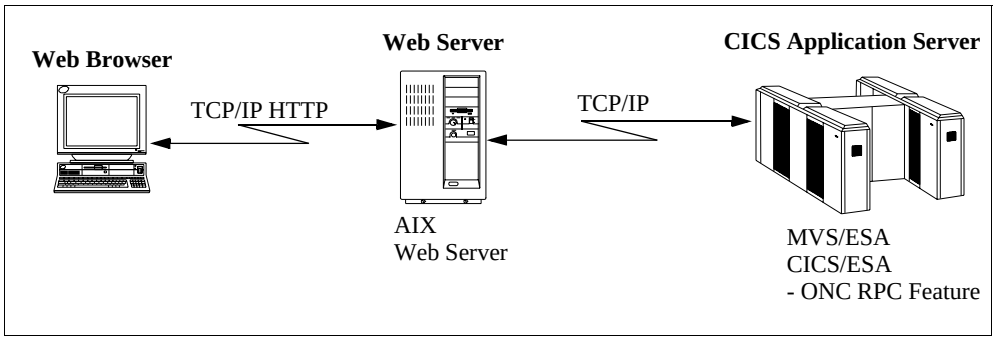


Figure 48. Connecting through the CICS ONC RPC Feature

5.2.4.4 MQ Series

If you have MQ series installed on your Web server, you can use MQ to route requests from your Web server to CICS/ESA by issuing MQ series calls from your CGI script. Figure 49 shows how you can connect a Web server to CICS/ESA using MQ Series.

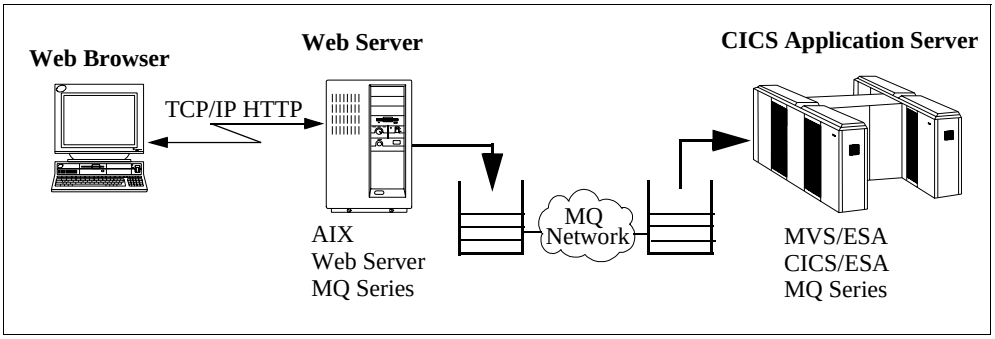


Figure 49. Connecting through MQ Series

IBM has announced its intention to provide an MQ Series Web Interface that will remove the need to code your own CGI script to issue MQ Series calls.

5.2.5 CICS Servers

You can use any of the following CICS server products to route your Web requests to CICS/ESA:

- CICS for OS/2
- CICS for Windows NT Version 2
- CICS/6000
- CICS for HP/9000
- CICS for DEC OSF/AXP
- CICS/400
- CICS/ESA
- CICS/MVS
- CICS/VSE

Note that CICS/ESA, CICS/MVS, and CICS/VSE do not support the CICS EPI; so the CICS Internet Gateway or applications using the EPI must be routed to CICS/ESA by way of an intermediary CICS server that does support the EPI.

5.3 Designing CICS/ESA Applications for the Web

When you write any CICS application, you must decide how that application interacts with the user. In the past, the user sat at a 3270 terminal (or more recently, a workstation emulating a 3270 terminal). Today, although use of fixed-function terminals such as the 3270 is in relative decline, the number of existing (sometimes called *legacy*) CICS applications that work with the 3270 interface remains very high. Consequently, interfacing with 3270 applications will remain important for some time to come. The CICS EPI was designed with this in mind, which is why the CICS Internet Gateway is a good first step in opening up CICS/ESA to the Web.

With the advent of the workstation, and distributed computing (of which the Web is just one aspect), new CICS/ESA application programs are being written not for the 3270 human interface, but to interface with another program. The task of managing the user interface, can, if desired, be delegated to a program running

on the workstation. It is for this style of application that the CICS ECI and other implementations of the RPC model are designed. They allow CICS/ESA programs to be invoked by means of a CICS distributed program link (DPL).

5.3.1 Accessing CICS/ESA 3270 Applications

The exercise of taking 3270 data streams and converting them to a different format is sometimes called *screen-scraping*. It is not a particularly elegant method of application programming; however, it is an essential tool to allow new non-3270 applications to access existing CICS/ESA 3270 applications that cannot easily be modified.

Fortunately, CICS provides a variety of tools to help you do this. There are currently three ways in that you can invoke a CICS/ESA 3270 application directly from a CGI script or GoServe filter:

- The CICS Internet Gateway
- The CICS Web Interface provided by CICS/ESA
- The EPI provided by the CICS workstation-based servers and CICS Client

5.3.2 HTML Awareness

Before you develop a CICS/ESA Web application, you must decide whether or not to make it "HTML-aware". You can:

- Delegate all the HTML-handling responsibilities to the invoking CGI script or GoServe filter, shielding your CICS/ESA application from the need to understand HTML. For example, a CICS/ESA application program that simply retrieves data from a CICS file or temporary storage queue, places it in the CICS COMMAREA, and returns it to the Web server does not need any knowledge of HTML or its syntax.
- Perform all or part of the HTML processing in your CICS/ESA application. An HTML-aware program dynamically builds an HTML document as it executes, perhaps using input supplied by the invoking CGI script to build the HTML.

Existing CICS/ESA 3270 applications that you access through the CICS Internet Gateway are examples of HTML-unaware applications, with all HTML being handled by the CICS Internet Gateway itself.

The two styles of programming are equally valid, and each has advantages and disadvantages:

Advantages of HTML-aware CICS/ESA Applications

- All the code for the application is in one place, so it is easier to maintain.
- Fewer interproduct or interplatform dependencies exist.
- HTML is becoming a universal tool in the client/server world. Making your CICS/ESA applications HTML-aware allows your CICS/ESA programmers to acquire new skills that are relevant for both Web and internal applications, and can be used on other platforms.
- Your CICS/ESA application can be the server for multiple client platforms.

If you a writing a large number of CICS HTML-aware applications, you may prefer to port the C utilities mentioned in 4.3, “Common Gateway Interface Scripts” on

page 39 to a generalized CICS/ESA application that can be linked to by applications needing to parse incoming forms data.

Disadvantages of HTML-aware CICS/ESA Applications

- Your CICS/ESA applications can be larger and more complex.
- You must restrict your application to one medium, in the same way that 3270 programs are restricted today. While HTML is likely to be around for a long time, this may be an unacceptable restriction.
- Application designers and programmers must have detailed knowledge of both the Web server and the CICS/ESA environment.

5.3.3 Using APPC or TCP/IP Sockets to Access CICS/ESA Applications

You can choose to use APPC or sockets calls to communicate between your Web server and CICS/ESA. If so, it is up to your Web server and CICS/ESA applications to manage communications between them. Figure 50 shows how you can connect a Web server to CICS/ESA using the CICS to TCP/IP for MVS Sockets Interface.

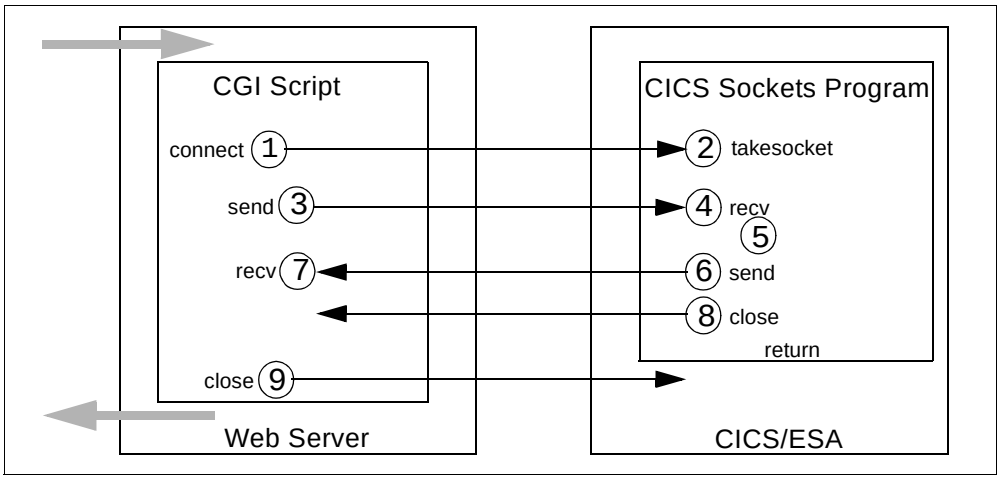


Figure 50. Connecting through TCP/IP for MVS Sockets Interface

A typical conversation might look like this:

1. The Web server establishes a connection with CICS/ESA.
2. The requested CICS/ESA application is started.
3. The Web server sends any input data to CICS/ESA.
4. The CICS/ESA application issues a RECEIVE for its input data.
5. The CICS/ESA application processes the request.
6. The CICS/ESA application issues a SEND to return any output to the Web server.
7. The Web server script issues a `recv` to read data returned from CICS.
8. The CICS/ESA and the Web server script close the connection.

While this is a more complex model for your application, it does give its components a greater level of control over the communication between them. For applications that are exchanging large amounts of data, it may well be more efficient to use this application programming model, because you can perform multiple SENDs and RECEIVEs on the same connection until all data has been exchanged.

See Chapter 10, “CICS Sockets Sample” on page 153 for a sample program illustrating how you can use TCP/IP sockets to communicate with CICS/ESA from your Web server.

5.3.4 Using DPL to Access CICS/ESA Applications

A Web server can use DPL in a variety of ways to invoke CICS/ESA transactions to process a request:

- CICS ECI
- CICS EXCI
- DCE RPC
- ONC RPC

Whatever the origin of the link to CICS/ESA, the mechanism used to pass data between the CICS/ESA program and the invoking program is exactly the same: on entry to the CICS/ESA program, it is passed to a communication area (COMMAREA) containing input data, and the CICS/ESA program passes back any output to the invoking program using the same COMMAREA.

In the case of the Web, the invoking program is the CGI script, and the data passed might be data that a user has entered into an HTML form on the Web browser.

The advantage of this approach is that, provided they use the same COMMAREA structure, your CICS/ESA application can be the server for multiple Web servers on a variety of platforms, using a variety of methods to link to CICS/ESA.

In our samples, we have concentrated on this application programming model. For details on these methods of accessing CICS/ESA, see the following references:

- For DCE RPC, see *MVS/ESA OpenEdition DCE Application Support: Programming Guide*.
- For EXCI, see *CICS/ESA External CICS Interface*.
- For details on the CICS ONC RPC Feature for MVS/ESA, contact your IBM representative.

5.4 Writing CICS/ESA Programs for the Web

The techniques used for CICS/ESA pseudo-conversational programming are very similar to those that are required for Web forms programming:

- Both are initiated by a user (at a Web browser or 3270 terminal).
- Both allow the program currently executing to specify which program is to run next.
- Both rely on being able to save information about the state of processing across invocations.

In this section, we look at how to apply CICS pseudoconversational techniques to Web transactions. Figure 51 on page 95 shows how you can use CICS facilities such as the COMMAREA to keep track of state within a typical CICS 3270 application.

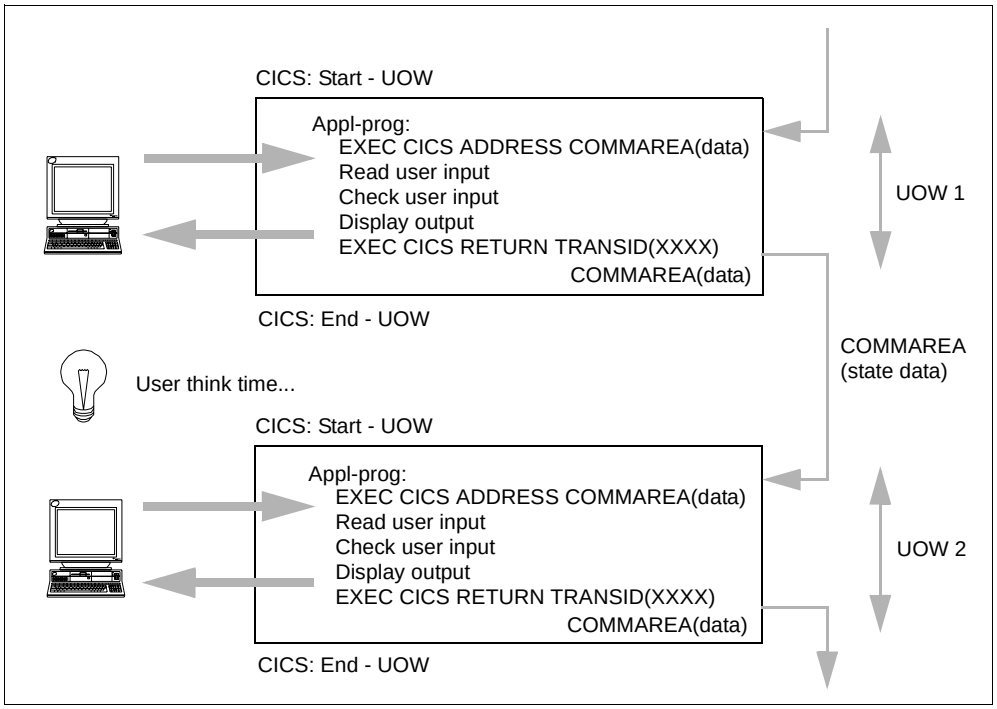


Figure 51. CICS Pseudo-conversational Processing

There are a range of issues familiar to CICS administrators and programmers that are not addressed by traditional Web clients and servers, and that must therefore be borne in mind when allowing Web access to CICS/ESA. The CICS pseudoconversational model is a well-tried method of communication between CICS and the user. Let us see how well we can apply this model to the Web environment, assuming that our CICS/ESA application is HTML-aware.

5.4.1 Pseudoconversation Initiation

In a traditional CICS pseudoconversation, users sitting at their 3270 terminal or workstation initiate a CICS/ESA transaction. The request is passed to CICS/ESA in the form of a 3270 data stream, or by way of an ECI call.

The Web environment is much the same. The user enters a URL, or updates an HTML form, and the CGI script that is invoked processes the request. In our case, the CGI script invokes a CICS/ESA application to process the request.

5.4.2 Passing Input Data to the Server

If the user is entering data at a 3270 terminal, at a CICS terminal on a CICS workstation, or via the EPI, then input data is passed to CICS/ESA in a 3270 data stream. If the application program is using BMS, the 3270 data stream is decoded by BMS before being passed to the application program.

If the CICS ECI is being used, any input parameters are passed to the CICS/ESA application in its COMMAREA.

A Web program receives any HTML forms input through the standard input stream, or in an environment variable, depending on the action specified on the

form. The Web server reads this data that can either be decoded by the Web server or sent to the CICS/ESA application unchanged.

5.4.3 Returning Responses from CICS/ESA

Once a CICS/ESA 3270 application is completed, it returns the results to the user in the 3270 data stream.

If a CICS/ESA program has been invoked using the ECI, the results are returned in the COMMAREA.

Any results to be returned to a Web user must be in HTML format when they arrive at the Web browser. Depending on how your application is structured, CICS/ESA can return HTML output to the Web server, or it can pass non-HTML output to the Web server and rely on the Web server to do the HTML formatting.

5.4.4 Terminating the Pseudo-conversation

For 3270 applications, the application itself terminates the pseudo-conversation by not priming CICS/ESA to continue it. How you do this depends on the design of the application. Typically, the application simply omits the TRANSID parameter when returning control to CICS.

In this respect, the Web transaction model is different. It is the user at the Web browser who determines whether the pseudo-conversation is at an end, for example by simply choosing not to invoke a particular server application again.

For ECI applications, the situation is effectively the same as for the Web. It is the client that ultimately determines whether the pseudo-conversation is at an end.

5.4.5 Specifying the Next CICS/ESA Program to Execute

A CICS/ESA 3270 application can specify in various ways the next transaction to be run for the pseudo-conversation. One is to make the first 4 bytes of the 3270 screen a hidden field containing the transaction identifier of the CICS transaction to be run.

We can use the same technique in a Web transaction by using a hidden field in an HTML form. In the case of a Web transaction, you may need to specify a program name rather than a transaction identifier, depending on the method being used to access CICS/ESA. The Web server can also elect to invoke a CICS/ESA server application different from that specified in the form if it so chooses.

For the ECI, the client specifies the next program to be run in the ECI parameter list.

5.4.6 Detecting Interruption to the Pseudo-conversation

For a CICS/ESA 3270 session, the situation is straightforward. If the 3270 session is lost, CICS/ESA detects the failure and raises the TERMERR condition. The 3270 application can handle the TERMERR condition, implementing any application recovery that is required, and CICS/ESA itself can restart the session using CICS/ESA recovery and restart facilities to check for lost data.

There is no equivalent function to this in the Web environment. There is no mechanism to allow a Web server to send unsolicited data to the Web browser. Recovery of the pseudoconversation can be initiated by the Web browser only by retrying the operation.

5.4.7 Data Integrity

For 3270 sessions, CICS/ESA provides recovery facilities to allow you to ensure that no data sent to or received from a 3270 terminal is lost. It is CICS/ESA that decides when to commit changes to its resources.

CICS clients using the ECI can manage their own data integrity by using the ECI_EXTENDED or ECI_NO_EXTEND parameters to control the scope of the logical unit of work.

Because it uses the Internet, the Web cannot provide the same level of data integrity between Web server and Web browser to which CICS administrators and programmers are normally accustomed.

With a Web application, you have at least two connections to consider:

- The link between your Web browser and your Web server
- The links between your Web server and CICS/ESA.

5.4.7.1 Data Integrity between Web Browser and Web Server

If your Web browser and Web server are connected across the Internet, data transmission between them is not guaranteed. If your Web browser issues a request and receives a response indicating that the function was completed successfully, you can be confident that the CICS/ESA transaction performed the function that you requested. If your Web browser does not receive a response, there is no way of knowing whether a CICS/ESA function that you requested has completed normally:

- It may not have run.
- It may have started and been completed normally, but the connection to the Web browser was lost before the results could be returned.
- It may have ended abnormally (abended).
- It may have been started and completed, but with the wrong result.

Although you will not normally be maintaining data resources on your Web browser, you can still design the CICS/ESA and Web server components of your application to ensure data integrity and consistency.

5.4.7.2 Data Integrity between the Web Server and CICS/ESA

The data integrity issues for the connection between the Web server and CICS/ESA are no different from those that arise for any other distributed CICS application.

If you are using the CICS family to manage communications between your Web server and CICS/ESA, then you can use the facilities provided by CICS to manage the logical units of work from your Web server. The CICS sample application in Chapter 8, “A Simple CICS Access Program: CICSWEB” on page 135 illustrates the use of the ECI to manage a logical unit of work that can span several calls to a CICS server program.

The level of data integrity depends on the communications protocol used to link the Web server and CICS/ESA. If your Web server and CICS/ESA are linked by a network using a communication protocol that allows you to manage your logical units of work, such as System Network Architecture (SNA) you can use the synch point facilities provided by APPC to ensure data integrity.

We recommend that, wherever possible, you manage your sensitive data from CICS/ESA. If you do need to share resources between CICS/ESA and your Web server, follow the established guidelines for CICS distributed processing.

5.4.8 Saving Information about the State of Processing

The Web was originally designed for "stateless" communication between client and server. It was only with the advent of forms programming that the need arose for an architected means of saving information about the state of processing across invocations of a Web server.

Although it is possible to save information about the state of processing in hidden fields in a form, this may not be ideal for the following reasons:

- If the information about the state of processing contains sensitive data, users may not want it to flow across the network, even in hidden fields.
- Where there are large amounts of information about the state of processing, it may be impractical to send it across the network each time the Web server needs to interact with the client.

A newly proposed draft IETF standard would allow state data to be stored in HTTP headers rather than in hidden fields in forms. Although this is a useful step forward, it still does not address the issue of having to send the information about the state of processing across the network. See the following URL for more information:

`ftp://ds.internic.net/internet-drafts/draft-kristol-http-state-info-00.txt`

You may therefore want a mechanism that allows you to save information about the state of processing that can be retrieved when new input is received for an ongoing Web conversation.

You first have to decide where you want to manage your information about the state of processing; this management can be done by your Web server CGI script, by your GoServe filter, or by CICS/ESA.

One method of minimizing the amount of information about the state of processing that has to flow across the network is to keep the information about the state of processing in storage on the Web server, and allocate a unique identifier to that information. You can then send the unique identifier in the form, so that when the forms input is received, the Web server can use the unique identifier to retrieve the information about the state of processing for that conversation.

You may decide to do some or all of your state management on the Web server (see Chapter 8, “A Simple CICS Access Program: CICSWEB” on page 135 for an illustration of how to do this). One of the advantages of doing at least some of the state handling on your Web server is that it can perform a routing function. If you have multiple CICS/ESA servers, you can put code into your CGI script or GoServe filter to analyze information about the state of processing returned by

the Web browser and invoke the appropriate CICS/ESA server for the requested function. This allows your Web server to provide a routing function similar to that provided by a CICS terminal-owning region (TOR) for conventional CICS transactions. Figure 52 on page 99 shows how you can maintain information about the state of processing data on your Web server when calling a CICS application.

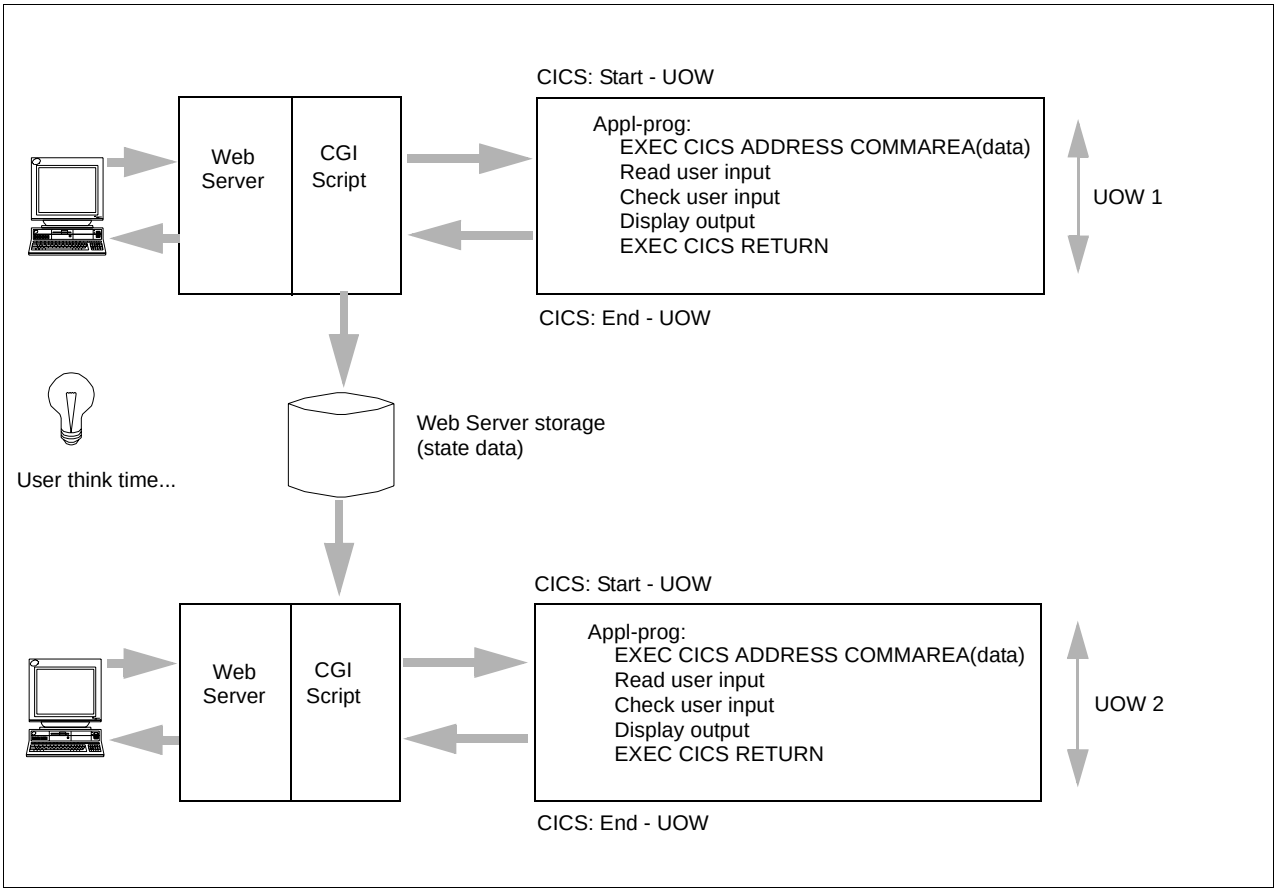


Figure 52. Web Server State Handling

The drawback of performing state management on your Web server is that the operating system environment may not be as well suited to this function as the CICS environment. State management is not a new problem for the traditional OLTP environment. CICS provides a variety of facilities that can be used to manage information about the state of processing for ongoing transactions, most of which are designed to make it easier to write pseudoconversational applications:

- Using CICS temporary storage is a common method of storing information about the state of processing. Naming conventions can be used to allocate unique identifiers to each CICS temporary storage queue. One of the most common techniques used to allocate unique identifiers is to incorporate the CICS terminal identifier in the temporary storage queue name.
- When a CICS terminal-oriented transaction ends, the next transaction to be run at that terminal can be specified either by the CICS program on its EXEC CICS RETURN statement, or in the CICS terminal definition for that terminal.

Managing information for the Web environment differs from managing information for the normal CICS environment in one important respect. In the traditional 3270 CICS environment, transactions are terminal-oriented. CICS associates information about the state of processing with a particular terminal, and many of the state management facilities that CICS provides are terminal-oriented. In the case of Web transactions that have been initiated using CICS DPL, no terminal is associated with the transaction, so these state management facilities cannot always be used. We therefore have to look at alternative methods of managing information about the state of processing for Web transactions (or for any other DPL or ECI application that does not have an associated terminal) that need to save information about the state of processing across transactions.

5.4.8.1 Allocating Unique Identifiers

The first problem you have when managing information about the state of processing for Web pseudoconversations is to find some method of allocating a unique identifier to a new pseudo-conversation. For 3270 terminals, CICS uses the terminal identifier to manage its internal information about the state of processing for the pseudo-conversation. Applications that use temporary storage to save their own information about the state of processing often use the terminal identifier to form part of the temporary storage queue name.

Because the Web pseudo-conversation has no terminal associated with it, you need an alternative naming scheme.

One possible alternative might be to use the IP address of the Web browser that originated the request to assign a unique identifier to the pseudo-conversation. However this only works if you know that:

- Only one pseudoconversation at a time will be running at a particular IP address, which is unlikely in a workstation environment.
- The request did not arrive via a SOCKS or proxy server, since requests from them reflect the address of the SOCKS or proxy server rather than the address of the originating browser itself.

It is therefore unlikely that a naming scheme using the IP address of the Web browser can be used for a generalized CICS/ESA server.

One way to resolve this problem is to implement a counter to allocate unique identifiers to pseudo-conversations. When a new pseudo-conversation is started, the counter is incremented, and the new number is allocated to the new pseudo-conversation. The counter must be held in generally accessible storage, either a temporary storage queue, or shared storage that can be addressed by any CICS/ESA program that needs it. Because this counter can be accessed concurrently by multiple CICS/ESA transactions, you need some kind of locking mechanism or compare and swap logic to prevent the same identifier from being allocated to two pseudo-conversations.

Once you have allocated a unique identifier to a Web pseudo-conversation, that identifier must be included in any form that is returned to the Web browser, so that subsequent input from the Web browser identifies which pseudo-conversation it should be associated with.

5.4.8.2 Timeouts

In the Web environment, users of Web browsers are often just surfing the Internet looking for information. Your pseudoconversation, having initialized its state storage, and returned a form to the Web browser, may never be invoked again, because the user of the Web browser found something more relevant, or more interesting to look at. Alternatively, the user may have completed the desired task using the pseudo-conversation, but not bothered to end the pseudo-conversation tidily. Having large numbers of idle Web pseudo-conversations on a busy CICS/ESA could lead to storage problems because of large amounts of storage being allocated to unused Web pseudo-conversations.

You therefore need some kind of timeout mechanism that allows you to purge Web pseudo-conversations that have been inactive for a set time. To do this you must do the following:

- Include time stamp information in your information about the state of processing.
- Write a CICS program that frees resources allocated to a pseudo-conversation that has been inactive for a set period of time.
- When designing your application, include something in your form to prompt the user to tell CICS/ESA when the pseudo-conversation is finished, so that any CICS/ESA resources associated with the pseudo-conversation can be released.

Your timeout program either can be run periodically to free resources from expired pseudo-conversations, or can be invoked each time a new identifier is allocated, or both.

5.4.8.3 Writing Your State Management Program

You can take two approaches when managing your Web pseudo-conversation information about the state of processing:

- Application-specific state management
- Generalized state management

Both approaches perform the same function, and are equally valid. If you are writing a relatively small number of Web transactions, you may prefer to incorporate your state management code into the application itself. If you are expecting to write a large number of Web applications, however, you may prefer to write a generalized routine that can be invoked by any Web application (or indeed any DPL application that has a need for information about the state of processing).

Application-Specific State Management (Topping and Tailing)

This approach to state management is sometimes called *topping and tailing* because it involves adding some extra code to the "top" (the start of the mainline path through the program) and to the "tail" (the end of the mainline path through the code); see Figure 53 on page 102.

```
Topping and Tailing
     Look at the input and look at the header
     If First Time (no conversation token)
     then allocate conversation token
     else restore state for conversation
          return input variables
     Remap commarea


     Save state for conversation x (for 10 minutes or timeout) or
     terminate (end of conversation)
     Here is my output for conversation x with template y
     This is my action tag for next program
```

Figure 53. Application-Specific State Management

Generalized State Management

This is the way we recommend that you handle your state management. See Figure 54 on page 102. It allows you to encapsulate all the state management code in one program, and if desired, to create standard transactions to monitor and control all the state resources in your CICS/ESA system. Your Web applications still need to be modified to call the state management program, but you do not have large amounts of code in each application devoted to state management. We provide a sample (see Chapter 9, “CICS State Management Program: CICSSTAT” on page 145) to show you how the basic functions can be coded.

```
Topping and Tailing
     Look at the input and look at the header
     If First Time (no conversation token)
     then call state manager to create pseudoconversation
     else call state manager to restore state for conversation
          return input variables
     Remap commarea

     If not last call then
     Call state manager to save state for conversation x
Else
     call state manager to terminate (end of conversation)
     Here is my output for conversation x with template y
     This is my action tag for next program
```

Figure 54. Generalized State Management

If you wish to make state management an integral part of your CICS/ESA system, you have the option of implementing it as a CICS/ESA task-related user exit (TRUE). This would involve some recoding of the sample in Chapter 9, “CICS State Management Program: CICSSTAT” on page 145, but would mean that you could make the handling of information about the state of processing even simpler, by creating a standard set of calls to be used by applications needing information about the state of processing management.

5.4.9 Data Conversion

In the 3270 data stream world, data conversion is not normally an issue since both the 3270 and the application deal with data encoded using EBCDIC. In a distributed computing environment, however, where you are using systems that hold and display numeric and character data in different formats, you are likely to be faced with the problem of data conversion. The Web is no exception. The CICS family of solutions provide facilities to make data conversion transparent to your CICS/ESA applications. The type of data conversion you need depends upon your application design.

If you use CICS/ESA only to retrieve and return preformatted HTML documents to a workstation Web server, you can avoid the need for data conversion altogether by storing the documents on CICS/ESA in ASCII format.

If you are using one of the CICS family products to communicate with CICS/ESA from the Web server, you can use the facilities provided by CICS/ESA to do your data conversion. See *CICS Communicating from CICS/ESA and CICS/VSE* for details. For illustrations of the use of these facilities, see 8.3, “CICSWEB Function Description” on page 141.

If you are using one of the non-CICS family methods of accessing your CICS/ESA applications, these usually also make the data conversion transparent to your application. DCE RPC and ONC RPC both provide mechanisms to convert data before it is passed to your CICS/ESA application. The CICS TCP/IP Sockets Interface for MVS/ESA Version 2.11 also provides a conversion utility for COBOL programs (see Chapter 10, “CICS Sockets Sample” on page 153).

5.5 CICS/ESA Systems Management Considerations

One of the advantages of enabling CICS to process Web requests is that you can take advantage of CICS facilities to address a range of systems management issues that may not be so easily handled on a normal Web server, such as:

- Routing of Web requests.
- Workload management.
- Security.
- Logging and auditing.

CICS/ESA provides various global user exit points you can use to perform these functions.

5.5.1 Routing of Web Requests

Once your Web request arrives at CICS/ESA, you can use standard CICS facilities to route the request within a CICSplex. The routing facilities available to you depend on how you are accessing CICS/ESA from the Web server. If you are using the CICS Internet Gateway, the EPI (through a CICS server that supports the EPI), or the ECI, then you have access to the full range of CICS transaction routing functions. See *CICS/ESA Dynamic Transaction Routing in a CICSplex* for details of how to configure your CICS/ESA system for transaction routing.

If you are using some other method of performing a distributed program link to your CICS/ESA applications, then you are more restricted in what you can do,

because you cannot necessarily control the transaction ID under which the program runs. You can however use the XPCREQ and XPCREQC global user exit points to perform a similar function.

If the CICS family is being used, XPCREQ is driven on both sides of the link, that is, in both the client and the server regions. You use XPCREQ to modify the SYSID at the time of the link request. One way to achieve this is by writing an application program to manage a list of SYSIDs in a global work area (GWA). The global user exit program accesses the GWA, and uses the information stored there to redirect the DPL request. XPCREQC is invoked after the link request is completed. You can use this exit to pass a response back to the application. For more details on the XPCREQ and XPCREQC global user exit points, and how to code them, see the *CICS/ESA Customization Guide.*

5.5.2 Workload Management

CICS/ESA Version 4.1 has workload management facilities that, when used in conjunction with the routing function described in 5.5.1, “Routing of Web Requests” on page 103, allow you to balance the workload in your CICSplex. Regardless of which release of CICS/ESA you are running, you need to review your workload management structure if you expect high volumes of Web transactions as a result of allowing Web access to CICS/ESA. For details, see the *CICS/ESA Performance Guide*.

5.5.3 CICS/ESA Security

Security has always been a principal concern in the traditional OLTP environment, so CICS/ESA already has significant security function built into it. The problem for CICS/ESA administrators who wish to give Web access to their CICS/ESA systems is how to use this security function to police requests coming in from the inherently less secure Internet environment. You need to consider a number of security implementation issues:

- Where do we put the firewall?
- Which component authorizes the request?
- How can we use CICS/ESA security functions?
- Which secure Web server do we use?

We discuss these questions in the following subsections.

5.5.3.1 Placing the Firewall

The placement of your firewall is crucial to the security of your system.

While firewalls can monitor flows in both directions, from the trusted network to the untrusted network and from the untrusted network to the trusted network, the emphasis is principally on allowing Web browsers inside the trusted network to access Web servers outside the firewall, while preventing users outside the firewall from accessing your trusted network. Proxy and SOCKS servers are designed to act on behalf of Web browsers, ***not*** on behalf of Web servers.

In this case, however, we need a security solution that allows Web browsers ***outside*** the firewall to access resources inside the firewall. If we put the Web server inside the firewall, there is no way of routing the Web browser request through the firewall to the Web server. We therefore must put the Web server

outside the firewall. Only then can it be accessed by Web browsers. Figure 55 on page 105 shows a configuration with the Web server outside the firewall.

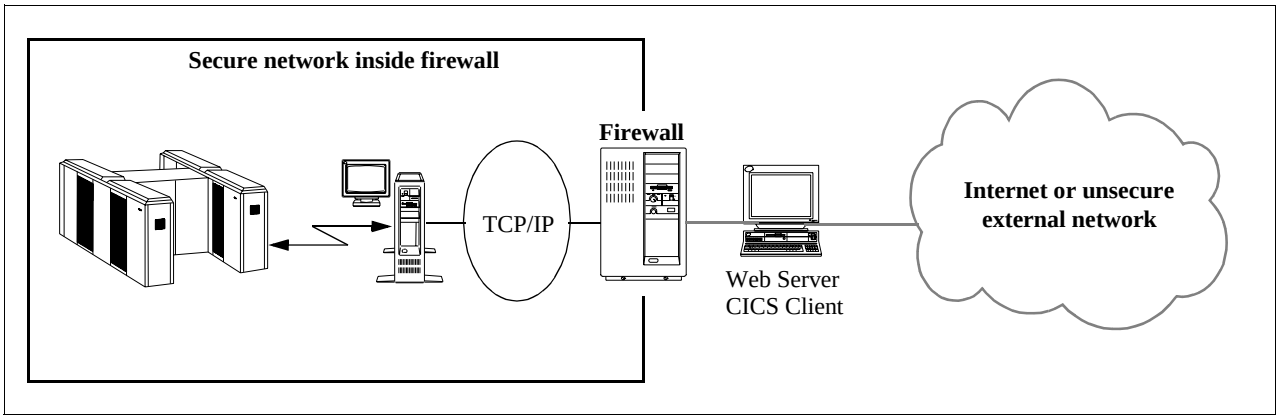


Figure 55. Connecting through a Firewall

Placing the Web server outside the firewall raises the question of how to communicate between the Web server outside the firewall, and the CICS system on the trusted network inside the firewall. You need to configure your firewall so that it will only allow flows into the trusted network between the Web server machine and selected CICS servers.

If you are using a CICS family solution, then the answer is to use the CICS Common Client to route requests from your Web server machine to a CICS system inside the firewall. This is a compromise solution, because we recommend that you have as little function as possible on your Web server. We strongly recommend that you do not place CICS server code on your Web server. Because using a firewall limits you to using TCP/IP to access CICS, the CICS system you use must be one of the CICS servers that support TCP/IP. Your Web browser request can then be routed to CICS/ESA from that server.

You can route requests directly from the Web server to CICS/ESA if you are using one of the following mechanisms that allow you to use TCP/IP for MVS to access CICS/ESA:

- CICS ONC RPC.
- DCE RPC.
- CICS to TCP/IP for MVS Sockets Interface.

5.5.3.2 Authorization

We recommend that you use the authorization facilities provided by your Web server to authorize incoming data. To use CICS/ESA security to add an extra level of security, then when the request is passed to the target CICS system, the Web server must already have mapped any authorization already granted to a CICS user ID and password.

5.5.3.3 Using CICS/ESA Security Facilities

If you are using the CICS Internet Gateway to link to existing 3270 applications, then you can sign on to CICS in the normal way.

Similarly, for the EPI and ECI, CICS provides you with facilities to manage user IDs and passwords. For illustrations of how CICS/ESA security can be used using the ECI, see Chapter 7, “Connectivity Tester: ECITEST” on page 125).

If you are using SNA or DCE to communicate with CICS/ESA then you can use the facilities they provide to manage your security.

If your configuration does not allow you to use the standard CICS/ESA, security facilities, then CICS/ESA Version 4.1 supplies some extra security-related EXEC CICS commands that you can issue from your CICS/ESA application:

- EXEC CICS VERIFY PASSWORD allows you to verify a user ID and password pair passed to your CICS/ESA application program by the Web server.
- EXEC CICS QUERY SECURITY allows you to verify whether the CICS/ESA application issuing the command has authority to access a CICS resource managed by the CICS external security manager, for example the Resource Access Control Facility (RACF).

For a detailed description of the above commands, see the *CICS/ESA Application Programming Reference*. The *CICS/ESA CICS-RACF Security Guide* provides detailed information on how to go about implementing CICS/ESA security.

5.5.3.4 Using Secure Web Servers

Products implementing security capabilities such as those described in Chapter 3, “Security” on page 15 are or soon will be available. For example, IBM recently introduced SSL and S-HTTP support in its Internet Connection Secure Server for AIX and OS/2 Warp, and Secure WebExplorer for OS/2 Warp. It will not be long before there are secure Web server and secure Web browsers available for all major platforms. As a long-term goal, we recommend that CICS system administrators who are considering allowing WWW access to CICS consider delegating some responsibility for authorization and encryption to this new generation of secure Web servers. CICS security can still be used as a second line of defense, once the Web server resolves the client request to a CICS user ID and password.

5.5.4 Logging and Auditing

To track the Web activity in your CICS/ESA system, you can use the HTTP header provided by the Web server as a useful source of information. Table 1 through Table 4 on page 38 shows the information available in HTTP headers. For examples of how to bring this information into your CGI script or GoServe filter, see Figure 10 on page 43 and Figure 11 on page 44. Once retrieved, this information can be passed to CICS/ESA either for use by the application itself, or for processing by a global user exit point such as XPCREQ.

Part 3. Sample Applications

Chapter 6. TestECI/TestEPI on CICS Gateway for Java

TestECI and TestEPI are Java programs to test the functionality of the CICS Gateway for Java. The programs source code is provided by the CICS Gateway for Java and can be found in the directory:

```
<install Dir>/JGate/java/ibm/cics/jgate/test.
```

TestECI and TestEPI are a good starting point to learn how to access CICS applications through the CICS Gateway for Java.

6.1 CICS Gateway for Java Scenario

As we described in Chapter 5.2.2, “The CICS Gateway for Java” on page 84, the gateway can run on a CICS server system or on CICS client system connecting to a CICS server. For the *TestECI* program it is not relevant on which operating system platform the CICS Gateway for Java is being installed. The program runs unaltered against all platforms. Following we describe the gateway configuration for OS/390 and AIX.

6.2 Configuring the CICS Gateway for Java on MVS

In this chapter we describe how to install and configure the CICS Gateway for Java on MVS.

6.2.1 Installation

The product files are packaged as a single compressed tar file. The tar format originates from Unix systems and was used to archive data onto magnetic tape. A program called tar is used to create and extract tar format archives. The package file is compressed to reduce its size and improve times for transfering from one system to another. The programs compress and uncompress can be used to work with compressed files. The installation is performed from an OpenEdition shell by invoking the commands to uncompress and extract the package into the OS/390 hierarchical file system (HFS).

These installation steps assume you have the file jg-11mvs.tarZ on your workstation. You will need a file system that allows for long file names, or rename the file to conform to your system‘s conventions. An example would be jg-11mvs.tgz for PC DOS systems that allow only eight characters for the name and three characters for the extension.

6.2.1.1 Getting the Package File to the HFS

First the package file must be placed into the HFS in a suitable directory. There are several ways you can do this depending upon the available setup at your site.

Using the file transfer protocol (FTP) program to an MVS dataset

From your workstation, use FTP to transfer the package file to the mainframe. The sequence of commands is shown in Figure 56 on page 110. Here the MVS dataset is created with the user‘s high-level qualifer prefixing it, but any dataset you have authority to create will do.

Attention: You must specify a binary file transfer!

```
stevel:/JavaGateway# ftp winmvs2c.hursley.ibm.com
Connected to winmvs2c.hursley.ibm.com.
220-FTPSERVE IBM MVS V3R2 at WINMVS2C, 23:07:10 on 1997/08/19
220 Connection will close if idle for more than 5 minutes.
Name (winmvs2c.hursley.ibm.com:slong): slong
331 Send password please.
Password:
230 SLONG is logged on.  Working directory is "SLONG.".
Remote system type is MVS.
ftp> bin
200 Representation type is Image
ftp> put jg-11mvs.tar.Z
200 Port request OK.
125 Storing data set SLONG.JG-11MVS.TAR.Z
250 Transfer completed successfully.
671417 bytes sent in 1.49 secs (4.4e+02 Kbytes/sec)
ftp> quit
221 Quit command received. Goodbye.
```

Figure 56. FTP to MVS

To place the package file into the HFS, you need to use the TSO command `oput`. Before using `oput`, make sure the directory in the HFS where the package file will go exists, in this case it is `/u/slong`. The binary option on the `oput` command is essential and the case of the second parameter must be correct. Figure 57 on page 110 shows the command on the TSO ready prompt.

```
READY
oput 'slong.jg-11mvs.tar.z' '/u/slong/jg-11mvs.tar.Z' binary
IGD103I SMS ALLOCATED TO DDNAME SYS00008
READY
```

Figure 57. oput Command in TSO

Using the Network File System (NFS)

The HFS directory where the package file is be placed can be mounted on the workstation using NFS. This allows a single workstation copy command to place the file in the HFS directory and is less error prone than using FTP. However, Your site may not have NFS configured or your workstation may not have the correct client software. If it is available, follow these steps to copy the package file to the HFS:

1. Mount the HFS directory on your workstation in binary mode. The command might look like:

   ```
   mount -t nfs winmvs2c.hursley.ibm.com/hfs/u/slong,binary /mountpoint
   ```

2. Copy the file to the mounted directory:

   ```
   cp /JavaGateway/jg-11mvs.tar.Z /mountpoint
   ```

Using FTP directly into the HFS

The FTP server on OS/390 can be configured to allow access directly to the HFS. This allows files to be transmitted into the HFS without having to use an MVS

dataset. Use the FTP program as before, but the oput command from TSO is not required. Figure 58 on page 111 shows a transcript of an FTP session to the HFS.

```
G:\JAVA>ftp wtsc52.itso.ibm.com
Connected to wtsc52.itso.ibm.com.
220-FTPD1 IBM MVS V3R2 at WTSC52.ITSO.IBM.COM, 22:48:32 on 1997-08-19.
220 Connection will close if idle for more than 5 minutes.
User (wtsc52.itso.ibm.com:(none)): cicsrs8
331 Send password please.
Password:
230 CICSRS8 is logged on.  Working directory is "/".
ftp> cd /u/java
250 HFS directory /u/java is the current working directory
ftp> bin
200 Representation type is Image
ftp> put jg-11mvs.tar.Z
200 Port request OK.
125 Storing data set /u/java/jg-11mvs.tar.Z
250 Transfer completed successfully.
631543 bytes sent in 10.50 seconds (60.18 Kbytes/sec)
ftp> quit
221 Quit command received. Goodbye.
G:\JAVA>
```

Figure 58. FTP Directly into HFS

6.2.1.2 Uncompressing the Package File

The file jg-11mvs.tar.Z is compressed using the Unix compress command. To uncompress it you use the uncompress command from an OpenEdition shell prompt. The command is:

```
CICSRS8@SC52:/u/java/: >uncompress jg-11mvs.tar.Z
CICSRS8@SC52:/u/java/: >
```

The uncompressed file is named jg-11mvs.tar, and the file with the .Z suffix is removed.

6.2.1.3 Extracting the Directories from the tar File

The remaining tar file must be expanded into a set of directories. (Figure 4 on page 117 shows the directory tree that is created when the tar file is expanded.) The command is:

```
CICSRS8@SC52:/u/java/: >tar -xopf jg-11mvs.tar
CICSRS8@SC52:/u/java/: >
```

The options on the tar command have the following effects:

- -x Expands the archive.
- -o Uses the owner and group information of the user when writing each file.
- -p Restores the original permission bits on each file.
- -f Precedes the name of the file to expand.

6.2.2 Configuration of the CICS Gateway for Java

Before the CICS Gateway for Java can be run, the startup script must be edited to reflect the CICS libraries on your system. The script is in the `/JGate/bin/mvs` directory and is called `JGate`. You can use the TSO command `oedit` to edit the file, or the OpenEdition editor `vi` from a shell. If the CICS Gateway for Java has been expanded in the directory `/u/java`, the TSO command to edit the script would be:

```
oedit '/u/java/JGate/bin/mvs/JGate'
```

At the bottom of the file, on lines 62 and 63, are the variables EXCI_OPTIONS and EXCI_LOADLIB that need to be changed. The values are the names of the data sets where CICS TS is installed on your system. You can set EXCI_OPTIONS if you have a data set that contains the EXCI options table DFHXCOPT. For example, if CICS TS has been installed into data sets that begin CICSTS12.CICS and the table DFHXCOPT is in CICSTS12.PROG.LOAD then the script should read:

```
EXCI_LOADLIB="CICSTS12.CICS.SDFHLOAD:CICSTS12.CICS.SDFHEXCI"
EXCI_OPTIONS="CICSTS12.PROG.LOAD"
```

The export statement in the script that follows these lines sets up the steplib of the CICS Gateway for Java. The steplib needs to reference the SDFHEXCI library for the EXCI modules, and the SDFHLOAD library for the module DFHTREX. DFHTREX is not necessary for the CICS Gateway for Java to run but it allows it to make trace entries that can be formatted with IPCS if a dump is taken.

When the changes have been made, save the `JGate` script file. This is all the configuration that is required for the CICS Gateway for Java to run.

6.2.3 Installing the DFHJAVA Group

One program definition needs to be installed on CICS TS in order to support the CICS Gateway for Java (MVS). This is DFHJVCVT and the definition is provided in the group DFHJAVA. By default, DFHJAVA is not included in the startup list DFHLIST, so you may like to add it to one of your own startup lists. You can also manually install the program definition or use autoinstall.

6.2.4 Configuring CICS Connection and Sessions

In order for an OS/390 program to use the EXCI to communicate with CICS TS, definitions for the connection and sessions need to be installed. The CICS Gateway for Java (MVS) can use a specific or generic connection. Refer to the EXCI users guide for a detailed description of how to define EXCI connections and sessions.

The sample group DFH$EXCI contains sample definitions that can be used by the CICS Gateway for Java (MVS). Installing this group creates a generic connection that will be used by CICS Gateway for Java (MVS) by default. It also creates a specific connection with the netname of BATCHCLI.

You can check that an EXCI connection exists with the CEMT transaction. After installing the DFH$EXCI group, you can call the `CEMT I CON` command.

6.2.5 Setting Environment Variables

Environment variables are associations of names with values that can be set up independent of programs that access them. Programs can read the values

associated with names and act upon them accordingly. The OpenEdition MVS Users Guide contains detailed descriptions of environmt variables and how they are set up. We describe three ways to set the variables that the CICS Gateway for Java (MVS) uses. Section 6.2.6, “Environment Variables Used by the CICS Gateway for Java (MVS)” on page 114 details the variables the CICS Gateway for Java (MVS) accesses.

6.2.5.1 The export Command

The OpenEdition export command can be used to set variables before the CICS Gateway for Java (MVS) is started up from a shell command line. For example see Figure 59

```
/u/java/JGate/bin/mvs: >export DFHJVPIPE=JVGATE1
/u/java/JGate/bin/mvs: >export DFHJVSYSTEM_00="SCSCPAA9-ITSO System TS 1.2"
/u/java/JGate/bin/mvs: >export DFHJVSYSTEM_01="BRANCH-Main branch server"
/u/java/JGate/bin/mvs: >JGate
CICS Gateway for Java, Version 1.1.3, 29H0948.
(C) Copyright IBM Corporation 1996. All rights reserved.
CCL6501I: Starting the CICS Gateway for Java with user specified values.
CCL6502I: [ Port = 2006 , Initial Connections = 1 , Maximum Connections = 100
,
CCL6502I:   Initial Workers = 1 , Maximum Workers = 100 ]
CCL6505I: Successfully created the initial Connection and Worker threads.
```

Figure 59. OpenEdition Export Command

6.2.5.2 Using Export in the JGate Script

To save having to type in the export commands each time you start the CICS Gateway for Java (MVS), you can place the statements in the JGate script file before the Java virtual machine is started. Figure 60 an example of the last few lines in the script file when some export commands are included.

```
export STEPLIB=${STEPLIB}:${EXCI_OPTIONS}:${EXCI_LOADLIB}

export DFHJVPIPE=JVGATE1
export DFHJVSYSTEM_00="SCSCPAA9-ITSO System TS 1.2"
export DFHJVSYSTEM_01="BRANCH-Main branch server"

#
# Start JGate
#
java ibm.cics.jgate.server.JGate $*
```

Figure 60. export in a JGate Script

6.2.5.3 Using the STDENV DD Name in the Startup JCL

In the JCL that runs the BPXBATCH program, you can code a DD card with the name STDENV. This can refer to inline data or a file that contains the name value pairs for the environment variables for the program that BPXBATCH starts. Here is how the same variables are coded in the JCL:

```
//STDENV DD *
```

```
DFHJVSYSTEM_00=SCSCPAA9-ITSO System TS 1.2
DFHJVSYSTEM_01=BRANCH-Main branch server
/*
```

6.2.5.4 Quotes in Environment Variable Values

When variables are defined with the export command, either on the command line or in the shell script, the value may need to be surrounded by quotes. If the value contains spaces or special characters then quotes are needed. In the example in Section 6.2.5.2, the DFHJVSYSTEM_nn variables need quotes because of the spaces, but DFHJVPIPE does not need quotes. In Section 6.2.5.3, quotes are not needed because the variables are set in JCL, not the script file.

6.2.5.5 Overriding Variables

The variables that take effect when the CICS Gateway for Java (MVS) is started are the final ones to be set. So if JCL is used to start the JGate script, any variables that are set in the script will override the variables of the same name in the JCL.

6.2.6 Environment Variables Used by the CICS Gateway for Java (MVS)

The CICS Gateway for Java (MVS) reads environment variables in order to obtain its customization options. The two options controlled by the variables are whether to use a specific or a generic EXCI connection, and what values to return for an ECIRequest.listSystems call. There are two variables that affect the operation of the CICS Gateway for Java (MVS).

6.2.6.1 DFHJVPIPE

In order for the CICS Gateway for Java (MVS) to use a specific EXCI connection, this variable must be set to the netname specified in the connection definition. If you are using the EXCI sample definitions in the group DFH$EXCI then a specific connection is installed called *EXCS*. In order that the CICS Gateway for Java (MVS) can use this connection, you must set the value of DFHJVPIPE to be BATCHCLI before starting up the gateway. If DFHJVPIPE is not set to any value, or is left unset, then the CICS Gateway for Java (MVS) will use the generic connection defined to CICS.

6.2.6.2 DFHJVSYSTEM_nn

You may set up to 100 variables of this form, with nn ranging from 00 to 99. The values are the names and descriptions of CICS systems to be returned in response to an ECIRequest.listSystems call. The value must be in the form of a string containing the name of a system followed by a hyphen, and then its description. For example,

```
/u/java: >export DFHJVSYSTEM_00="SCSCPAA9-ITSO System TS 1.2"
/u/java: >export DFHJVSYSTEM_01="BRANCH-Main branch server"
```

If you start up the CICS Gateway for Java (MVS) after issuing these commands and use the TestECI program, the output is as shown in Figure 61 on page 115.

```
TestECI - simple test of CICS Gateway for Java functionality
=== Test Parameters ===
CICS Gateway : wtsc52.itso.ibm.com:3006
ECI Server   : null
ECI UserId   : null
ECI Password : null
No of programs given : 0
=== Connect to Gateway ===
Successfully created JavaGateway
=== Available Servers ===
System : SCSCPAA9, Description : ITSO System TS 1.2
System : BRANCH, Description : Main branch server
Successfully closed JavaGateway
```

Figure 61. TestECI output on MVS

6.2.7 Running the CICS Gateway for Java (MVS)

The CICS Gateway for Java (MVS) can be started from an OpenEdition shell prompt or by submitting JCL that runs the BPXBATCH program. BPXBATCH is an OS/390-supplied program that executes OpenEdition programs and shell scripts that reside in the HFS. Figure 62 on page 115 shows some JCL that can be used to run the JGate script file. It is assumed that the CICS Gateway for Java (MVS) was installed in the /u/java/JGate directory.

```
//JGATE    JOB (999,POK),'JGATE',CLASS=A,MSGCLASS=T,
//            NOTIFY=&SYSUID
//BPXJGATE EXEC PGM=BPXBATCH,
//           PARM='SH /u/java/JGate/bin/mvs/JGate
//                     -noinput',
//           REGION=28M
//STDIN  DD PATH='/dev/null',
//          PATHOPTS=(ORDONLY)
//STDOUT DD PATH='/u/java/JGate/jgateo.log',PATHOPTS=(OWRONLY,OCREAT),
//          PATHMODE=SIRWXU
//STDERR DD PATH='/u/java/JGate/jgatee.log',PATHOPTS=(OWRONLY,OCREAT),
//          PATHMODE=SIRWXU
//STDENV DD *
DFHJVSYSTEM_00=SCSCPAA9-ITSO System TS 1.2
DFHJVSYSTEM_01=BRANCH-Main branch server
```

Figure 62. JCL to run JGate Script on MVS

Since BPXBATCH does not support MVS files for its standard output and standard error, you need to specify HFS files. In Figure 62 on page 115 these are `/u/java/JGate/jgateo.log` and `/u/java/JGate/jgatee.log`. In order to view these it is necessary to use the TSO OEDIT command or enter an OpenEdition command shell and use the OpenEdition file manipulation commands.

6.2.7.1 Using nohup for Background Execution

When the CICS Gateway for Java (MVS) is started from an OpenEdition command line such as OMVS, the user cannot log out of OpenEdition without

stopping the gateway. To start the CICS Gateway for Java (MVS) as a background program that does not stop when the user logs out, use the command shown in Figure 63.

```
/u/java/JGate/bin/mvs: >nohup JGate -noinput &
[1]     67108868
/u/java/JGate/bin/mvs: >nohup: sending output to "nohup.out"
/u/java/JGate/bin/mvs: >
```

Figure 63. Starting JGate as "no login" User

This causes all the output to go to the file `nohup.out`, which can be viewed at any time. To stop the CICS Gateway for Java (MVS) at a later date, it is necessary to use the `kill` command. At a later date, to find the number of the job to kill, use the `jobs` command first. The commands Figure 64 stop the CICS Gateway for Java (MVS) when it has been started using the `nohup` command. Finally the `cat` command displays the log file.

```
/u/java/JGate/bin/mvs: >jobs
[1] + Running nohup JGate -noinput &
/u/java/JGate/bin/mvs: >kill %1
[1] + Done(143) nohup JGate -noinput &
/u/java/JGate/bin/mvs: >cat nohup.out
CICS Gateway for Java, Version 1.1.3, 29H0948.
(C) Copyright IBM Corporation 1996. All rights reserved.
CCL6501I: Starting the CICS Gateway for Java with user specified values.
CCL6502I: [ Port = 2006, Initial Connections = 1 , Maximum Connections = 100
,
CCL6502I:   Initial Workers = 1 , Maximum Workers = 100 ]
CCL6505I: Successfully created the initial Connection and Worker threads.
CEE5205S The signal SIGTERM was received.
[1] + Done(143) ?
  83886115      FSUM7750 Terminated     java
/u/java/JGate/bin/mvs: >
```

Figure 64. Stopping JGate with the kill Command

6.3 Configuring the CICS Gateway for Java on AIX

In this section we describe how to install and configure the CICS Gateway for Java on AIX. The code may be downloaded for free on:

`http://www.hursley.ibm.com/cics/internet/cicsgw4j/jgdown.html`

6.3.1 Installation

Make sure that you login as user root and that you have enough disk space left in the file system you like to install the CICS Gateway for Java. Now follow these steps:

1. The downloaded file is a UNIX compressed tar file and has to be moved to the directory where you will create the root directory for this installation. For example,

```
# mv /tmp/jg-11aix.tar.Z /usr/lpp
# cd /usr/lpp
```

2. Now uncompress the file jg-11aix.tar.Z

```
# uncompress jg-11aix.tar.Z
```

3. Expand the uncompressed file jg-11aix.tar

```
tar -xvf jg-11aix.tar
```

4. This should create the root directory JGate, and the directory structure underneat as shown in Figure 65.

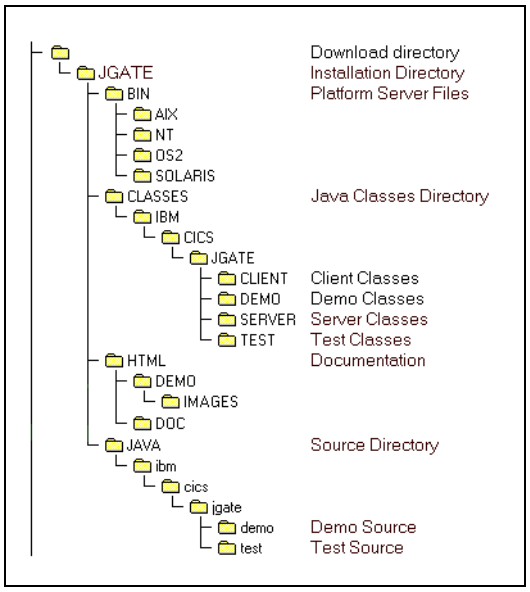


Figure 65. CICS Gateway for Java Directory Structure on AIX

6.3.2 Configuration

Define to your Web server the location of the root directory into which you have installed the CICS Gateway for Java code on AIX. Your particular Web server documentation will explain how to do this. In our case, we used the IBM Internet Connection Secure Server Version 4.2 for AIX, and we simply made a symbol link from the JGate root directory to the Web servers pub directory:

```
# cd /usr/lpp/internet/server_root/pub
# ln -s /usr/lpp/JGate JGate
```

This is necessary in order to be able to point from a Web page to the JGate classes.

However, this is a quick way to get access to JGate through the Web server. In order to gain any security, you must customize the httpd configuration file and add the JGate directory structure as an accessible path according to the manual. There is also a very good redbook available on customizing the IBM Internet Connection Secure Server, SG24-4805-00.

Add the JGate classes to the CLASSPATH variable of your environment. In our example,

```
# export CLASSPATH=$CLASSPATH:/usr/lpp/JGate/classes
```

We recommend adding this statement permanently into the `/etc/environment` file.

6.3.3 Starting the CICS Gateway for Java on AIX

You must start the CICS Gateway for Java at the AIX operating system command prompt of the computer on which you have installed it. You can use the default start options or you can set up user-defined options during startup. To work with the default startup options, you first must set your working directory to the place where the JGate program is. In our example,

```
# cd /usr/lpp/JGate/bin/aix
```

Then press `JGate` at the command prompt and press Enter. You will see the following output shown in Figure 66.

```
CCL6500I: Starting the CICS Gateway for Java with default values.
This will be followed by two lines showing the values which are being used:
CCL6502I: [ Port = 2006 , Initial Connections = 1 , Maximum Connections = 100
,
CCL6502I:   Initial Workers = 1 , Maximum Workers = 100 ]
```

Figure 66. Output from JGate on AIX

To override the defaults, simply type `JGate` at the command prompt and add some or all of the following options:

- -port=<port_number>
  TCP/IP port number to listen on
- -initconnect=<number>
  Initial number of Connection Manager threads
- -maxconnect=<number>
  Maximum number of Connection Manager threads
- -initworker=<number>
  Initial number of Worker threads
- -maxworker=<number>
  Maximum number of Worker threads
- -trace
  To enable extra tracing messages
- -time
  To enable timing information in messages
- -noinput
  To disable the reading of input from the console

We worked with the CICS Gateway for Java V1.1.2, which had the restriction that no concurrent ECI and EPI requests were supported. In order to support concurrent ECI and EPI requests, we had to start two different JGate programs, assigned to different TCP-IP ports, with the `-port=<port_number>` option. This problem was described in the JGate readme file and may be solved in the future.

6.3.4 Stopping the CICS Gateway for Java

If you have startet the JGate with the `-noinput` flag set, you must use the UNIX `kill` command to stop the gateway on AIX. To do so, you must first evaluate the process number of JGate with:

```
# ps -ef | grep -v grep | grep JGate
```

Afterward use the `kill` command against the process number:

```
# kill <JGate process number>
```

If you did not use the -noinput flag to start JGate, simply enter `Q` at the command line and press Enter to stop it.

6.4 TestECI

TestECI is a sample program to test the functionality of the CICS Gateway for Java. With ECITest, you can connect to a CICS Gateway for Java and send an ECI request to a CICS server. On this CICS server, you can start one or even more CICS programs which all run in the same UOW. ECITest cannot pass a COMMAREA to the called programs.

In our installation, we placed the source of ECITest in `/usr/lpp/JGate/java/ibm/cics/jgate/test` as illustrated in Figure 4 on page 117.

6.4.1 Running TestECI

TestECI can be used as either Java a application or as an applet. If you use it as a Java application, you must start it with the code shown in Figure 67.

```
java ibm.cics.jgate.test.TestECI    [jgate=jgate_server]
                                    [jgateport=jgate_port]
                                    [server=cics_server]
                                    [userid=cics_userid]
                                    [password=cics_password]
                                    [prog<0..9>=prog_name]
                                    [status]
                                    [trace]
```

Figure 67. TestECI Structure

Where:

- `jgate_server` is the address of the Gateway to connect to.
- `jgate_port` is the Gateway port on jgate_server (defaults to 2006).
- `cics_server` is the name of the CICS server to receive ECI requests.
- `cics_username` and cics_password are the userID and password.
- `prog_name` is the name of a CICS server program. You can specify up to ten program names.
- `status` will cause the program to query the status of all the known CICS servers.
- `trace` will cause tracing information to be produced.

If you use TestECI as an applet, you must incorporate an applet tag into the HTML page where you place the ECITest output. Figure 68 on page 120 shows an example of such an implementation:

```
.
.
.
<h1>IBM CICS Gateway for Java - TestECI sample </h1>
<p>TestECI is a sample program which allows you to test the
functionality of the CICS Gateway for Java.
With TestECI you can connect to a Gateway and then
send one or more ECI requests to a CICS server.
<p>
Further information on TestECI is available as part of the
<a href="/usr/lpp/JGate/html/doc/jgclass.html#testeci">CICS Gateway for Java
documentation</a>

<table><tr><td>
<br>
<hr>
<applet
codebase="/usr/lpp/JGate/classes"
code="ibm.cics.jgate.test.TestECI.class" width=520 height=290>
    <param name=jgate value=sinop>
    <param name=jgateport value=2006>
    <param name=server value=cicssj01>
    <param name=userid value=guest>
    <param name=password value=guest>
    <param name=prog0 value=DFHCSSF0>
    <param name=status value=yes>
    <param name=trace value=yes>
<blockquote>
<hr>
<em>
Sorry, you are viewing this page with a browser
that doesn't understand the APPLET tag.
<br>
You need to upgrade to a Java-enabled browser.
<br>
You can download the latest Netscape browser from
<a href="http://home.netscape.com"> http://home.netscape.com </a>
</em>
.
.
.
.
```

Figure 68. Applet tag for TestECI

If this HTML page is viewed with a Java-enabled browser, the TestECI applet runs the following procedure:

1. The Java program creates an instance of a ibm.cics.jgate.client.JavaGateway object.
2. The Java program creates an instance of a ibm.cics.jgate.client.ECIRequest object containing the request that it wishes to make.

3. The Java program then flows the request to the CICS Gateway for Java using the flow method of the JavaGateway object.
4. The Java program checks the return code of the flow operation to see whether the request was successful.
5. The Java program then closes the JavaGateway object.

Figure 69 on page 121 shows the output of a successful finished TestECI transaction. All included programs (prog0...9) run in the same UOW.

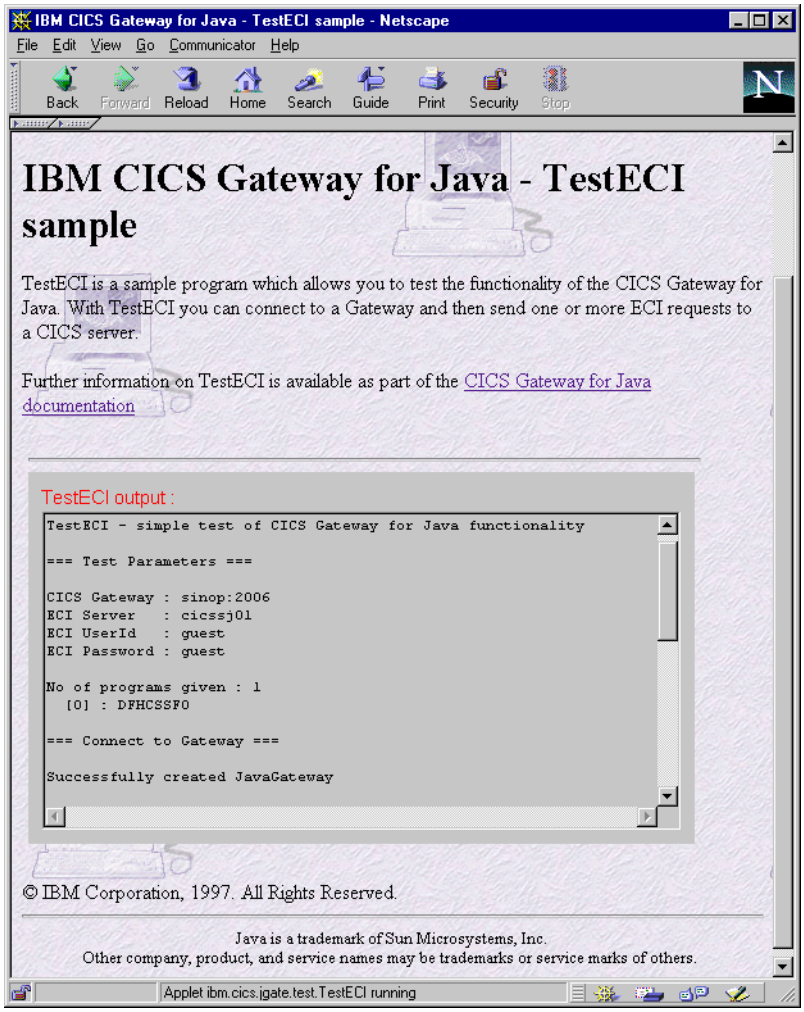


Figure 69. Output of the File TestECI.html

6.5 TestEPI

TestEPI is a sample applet that also allows you to test the functionality of the CICS Gateway for Java. With TestEPI you can connect to a Gateway and then send one or more EPI requests to a CICS server.

TestEPI uses two other classes: RequestDetails and EPIStrings which are also provided with the sample code.

6.5.1 Running TestEPI

TestEPI has a scrolling list that contains two EPI requests:

- List CICS servers
- Run transaction CECI

The TestEPI applet tag in Figure 70 on page 122 has been also placed into an HTML page similar to the TestECI sample, however; all needed parameters are being asked by the applet itself and not through the applet parameter list.

```
.
.
.
<h1>IBM CICS Gateway for Java - TestEPI sample </h1>
<p>
TestEPI is a sample applet which allows you to test the functionality of
the CICS Gateway for Java. With TestEPI you can connect to a Gateway and then
send one or more EPI requests to a CICS server.
<p>
Further information on TestEPI is available as part of the
<a href="/usr/lpp/JGate/html/doc/jgclass.html#testepi">CICS Gateway for Java
documentation</a>

<table><tr><td>
<br>
<hr>
<applet
codebase="/usr/lpp/JGate/classes"
code="ibm.cics.jgate.test.TestEPI.class" width=650 height=500>
<blockquote>
<hr>
<em>
Sorry, you are viewing this page with a browser
that doesn't understand the APPLET tag.
<br>
You need to upgrade to a Java-enabled browser.
<br>
You can download the latest Netscape browser from
<a href="http://home.netscape.com"> http://home.netscape.com </a>
</em>
.
.
.
.
```

Figure 70. Applet Tag for TestEPI

Figure on page 123 shows the output of a successfull finished TestEPI transaction.

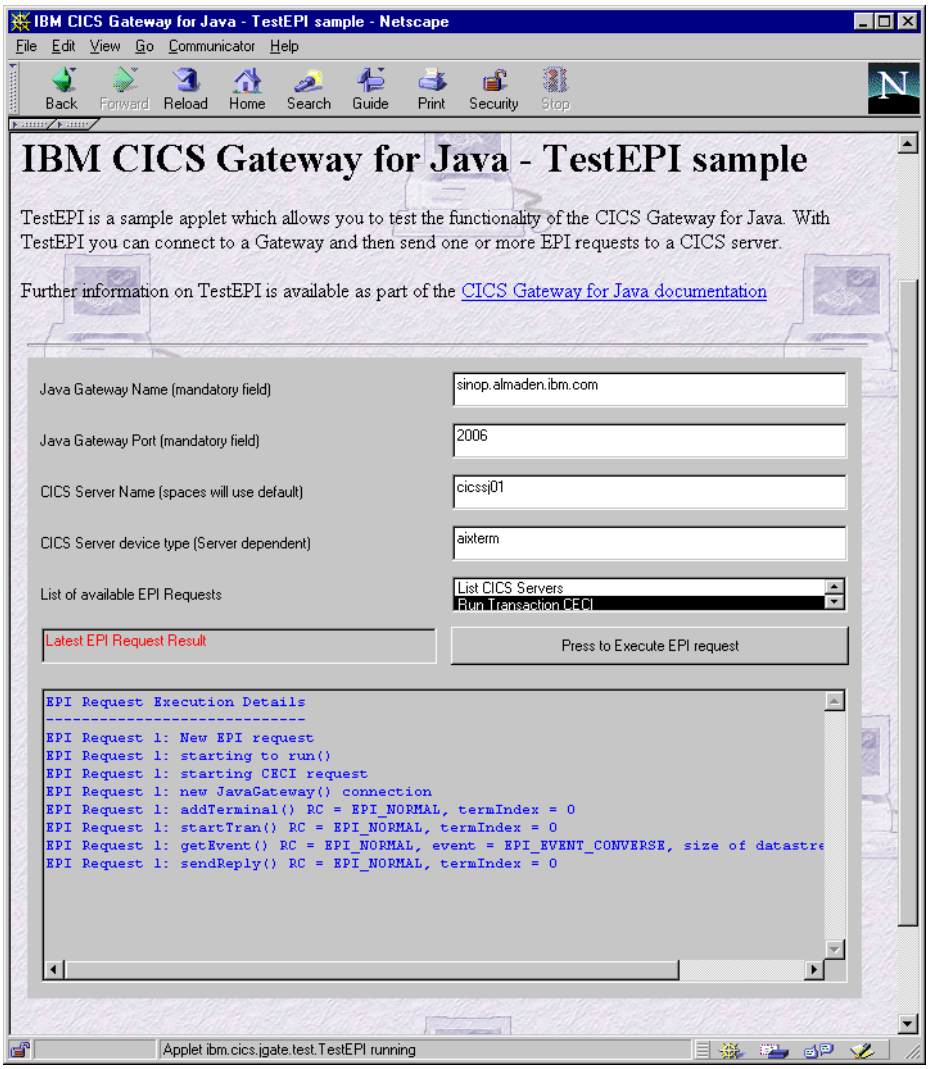


Figure 71. Output of the File TestEPI.html

Chapter 7. Connectivity Tester: ECITEST

In this chapter we describe ECITEST, a simple application that illustrates a number of basic building blocks you need to access a CICS application from the Web. Use ECITEST to call a CICS program from a Web server using the CICS ECI.

Source code for ECITEST components is included in Appendix A, “ECITEST Source Listings” on page 157.

7.1 What Does ECITEST Do?

ECITEST performs two basic functions: First, it prompts you to enter a CICS user ID and password, as shown in Figure 72.

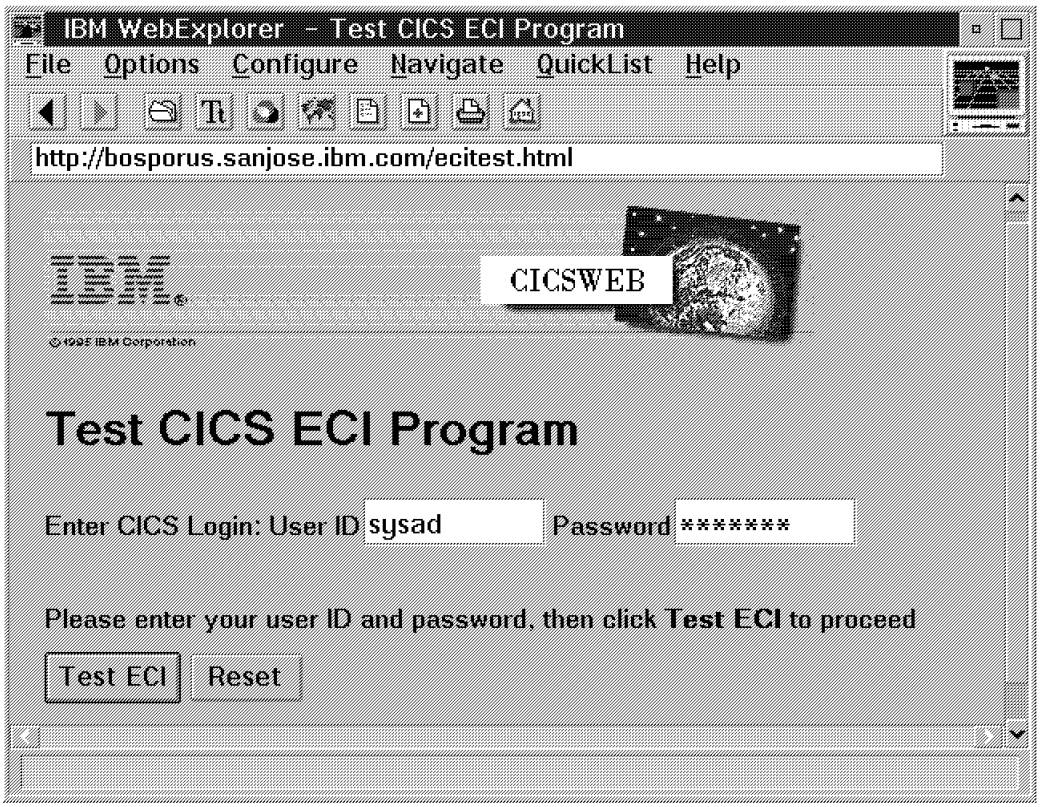


Figure 72. ECITEST Sample Application: Login Page

ECITEST then presents you with a page that contains a form with two input fields, as shown in Figure 73 on page 126.

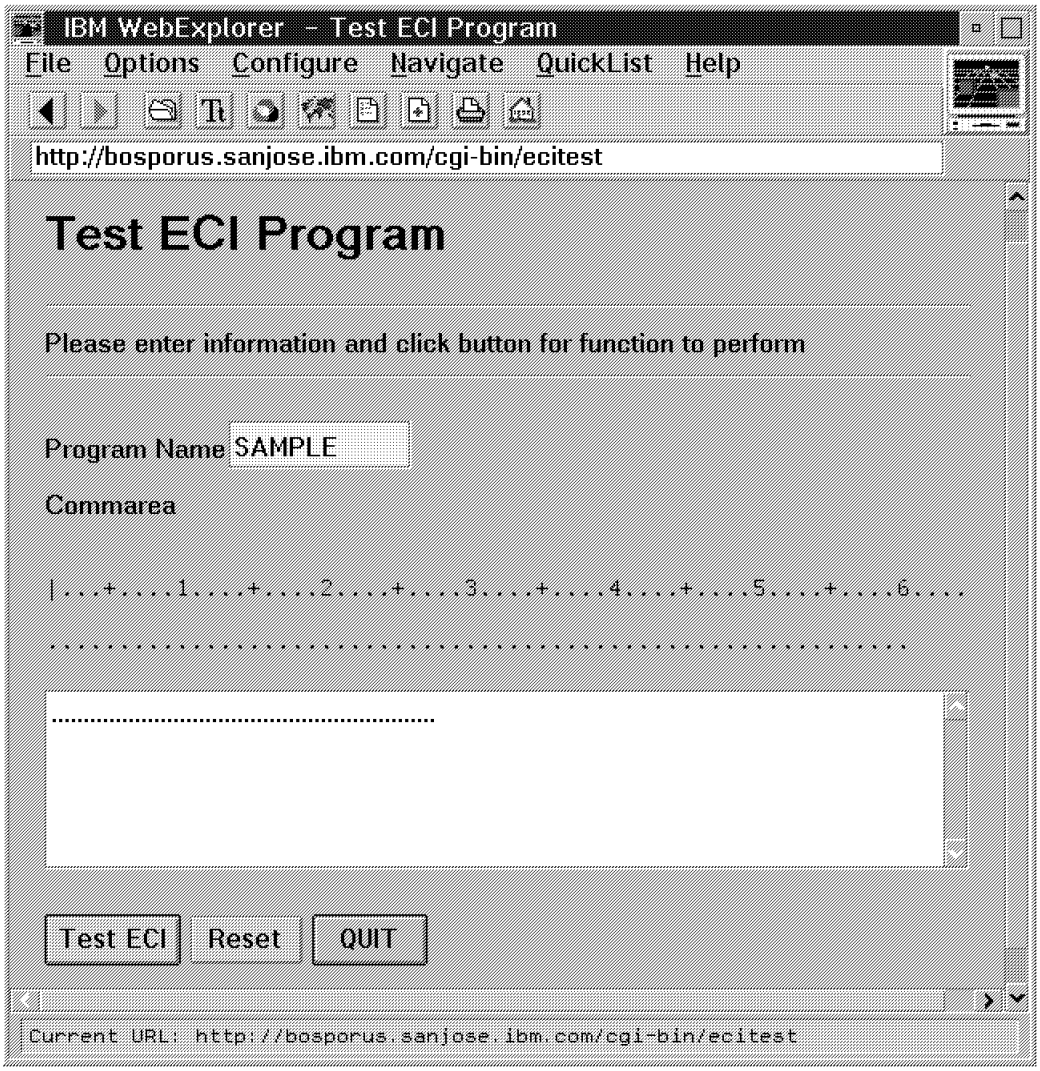


Figure 73. ECITEST Sample Application: Data Entry Initial Display

Enter the name of a CICS program in the first field, and the contents of the CICS COMMAREA in the second field. When you press Enter, or click on the Test ECI button on the form, ECITEST calls the program you specified.

After calling your CICS program, ECITEST redisplays the form, including any data that it returns in the CICS COMMAREA, as shown in Figure 74 on page 127.

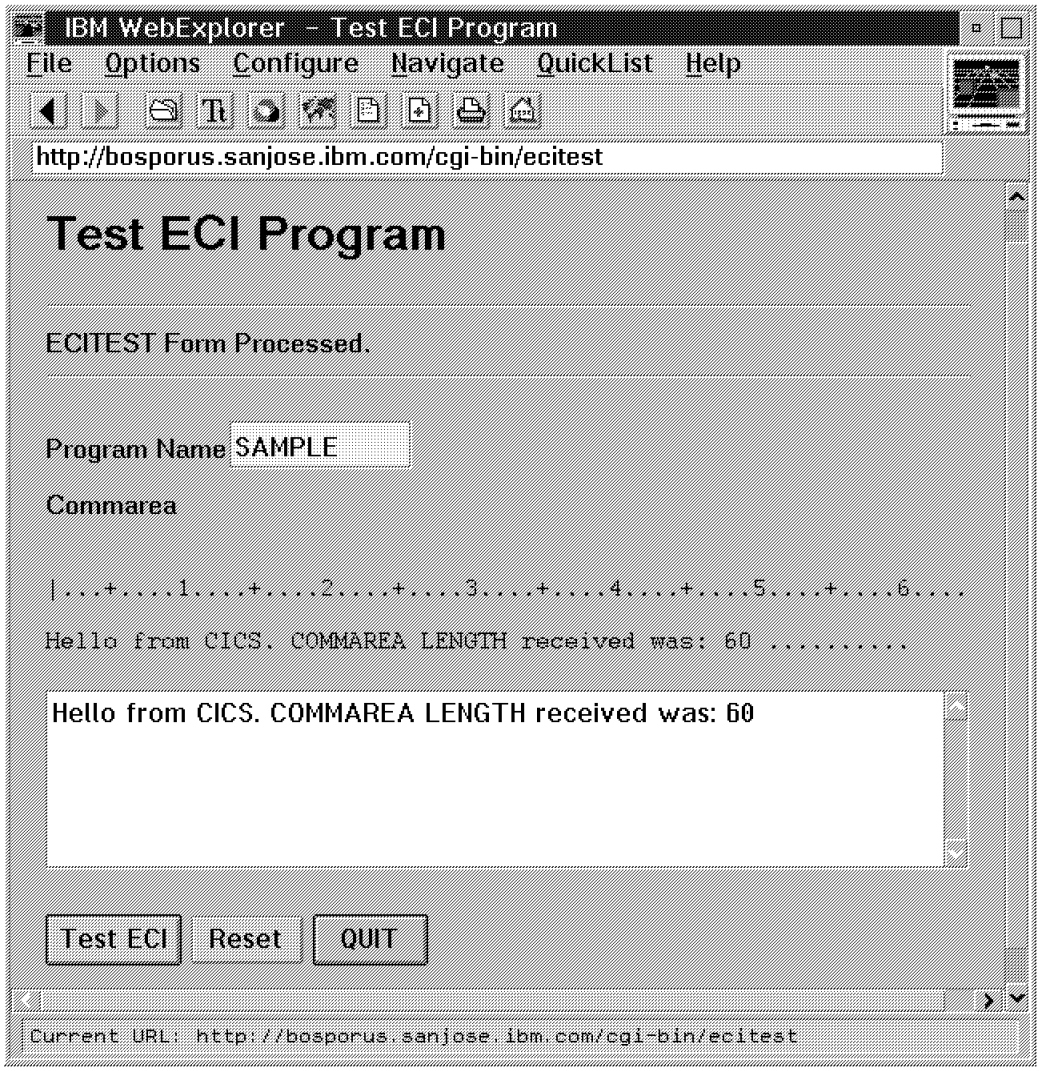


Figure 74. ECITEST Sample Application: Data Entry Program Response

You can then repeat this process to call the program with a different COMMAREA, or call a different program altogether. ECITEST also displays a message just below the page title telling you the result of the call. If an error condition such as a CICS application abend is detected, the message includes information about the error.

7.2 ECITEST Components and Interfaces

ECITEST consists of two main components:

- A Web server extension, or gateway, that provides the link between the Web environment and CICS
- A CICS program that performs the requested application function.

ECITEST also requires that you have a Web browser to provide the user interface to the functions that ECITEST provides.

These ECITEST application components use the following interfaces:

- The Web browser communicates with the Web server using the HTML data stream and HTTP protocols.
- The Web server communicates with the ECITEST Web server extension using the mechanisms defined by the Web server. We provide examples of two interfaces in this book: the Common Gateway Interface (CGI) used by the IBM Internet Connection Server and several other Web servers, and the filter interface provided by GoServe. In both cases, ECITEST uses a Web server extension written using REXX.
- The Web server extension communicates with CICS using the CICS ECI.
- The CICS ECI communicates with the destination CICS system on which the CICS program runs. This covers a number of possible CICS connectivity scenarios, ranging from an ECI call to a CICS program on CICS OS/2 or CICS/6000 running on the same workstation as the Web server, to an ECI call that is routed across a multiprotocol network through one or more intermediate CICS systems to a CICS/ESA system. Section 5.1, “Connecting CICS to the Internet” on page 78 includes information about CICS connectivity options you can use.

ECITEST processing is distributed over three platforms: A workstation with a Web browser, a server running a Web server, and a CICS host. While nothing prevents two or even all three components from running on the same physical machine, this is not a typical setup except, perhaps, while you are developing and testing your applications. Figure 75 shows the relationship of these three platforms.

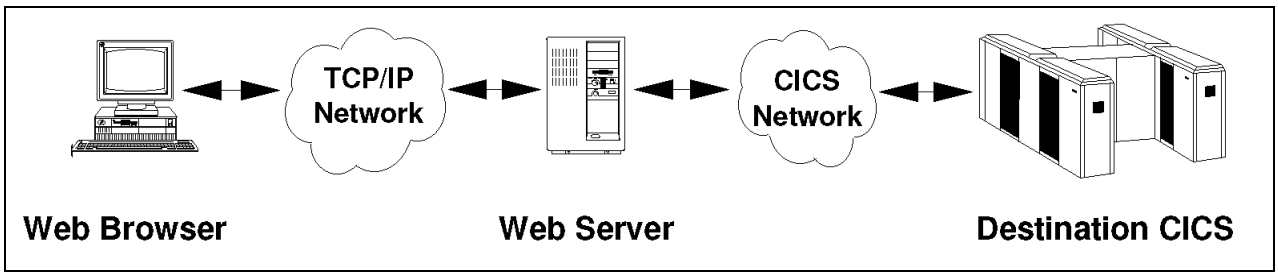


Figure 75. ECITEST Components and Interfaces

Clearly the Web server platform is the key to linking the Web and CICS worlds. Figure 76 shows the connection between Web and CICS functions running on the Web server.

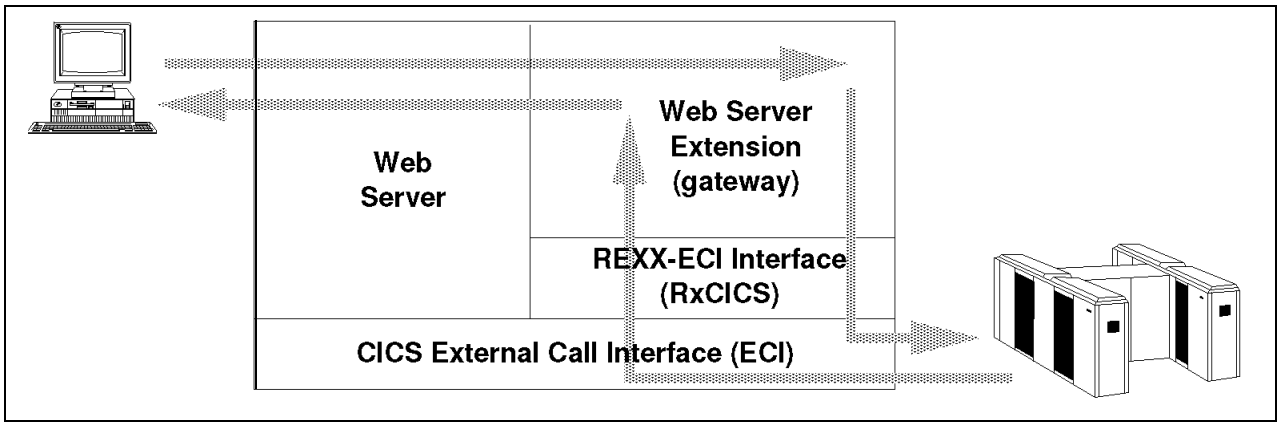


Figure 76. Web to CICS: Web Server Components

7.3 ECITEST Function Description

Functions that ECITEST must perform include:

- Invoking an application-specific Web server extension or gateway
- Obtaining user input from the Web browser
- Maintaining information about the state of processing
- Passing data to and from CICS
- Generating dynamic HTML documents
- Deleting information about the state of processing on the Web server.

We look at how ECITEST performs each of these functions as an example of how you can perform these functions in your own applications.

7.3.1 Invoking an Application-Specific Web Server Extension or Gateway

The mechanism you use to tell the Web server to start your application depends on the server. The most common method is that defined by the Common Gateway Interface (CGI) specification, which is implemented by a number of servers including the Domino Go Webserver.

We also look at how you invoke an application using the GoServe Web server for OS/2.

7.3.1.1 Invoking the Web Server CGI Script

When using the CGI, specify the application extension or script to be executed in the URL. For example, a URL of `http://servername/cgi-bin/myprog/` requests that the server invoke program `myprog`. The string `cgi-bin` after the server name in the URL is the key you use to trigger execution of your script. You can customize most servers to use other strings to trigger scripts as well.

`Note that myprog` can be any executable program supported by the operating system under which the server is running. For example, on OS/2 Warp, you could write your script using C or C++, or use OS/2 command files using REXX. We chose to use REXX because its powerful string handling facilities are ideally suited to writing Web server scripts.

Any data in the URL appearing after the script name is not used by the server, but is made available to the CGI script so you can use it to control the operation of your application if you wish.

There are several ways in which the Web browser can generate the URL that triggers your script:

- The user can type the URL manually using the Web browser's command line.
- You can specify the URL as a hypertext link in an HTML document that the user displays, for example,

  ```
  <A HREF="http://bosporus.sanjose.ibm.com/cgi-bin/cicsweb/cicsweb.html">
  cicsweb.html</A>
  ```

  If your CGI script sent the document, you could also specify a relative URL, for example,

  ```
  <A HREF="cicsweb/cicsweb.html">cicsweb.html</A>
  ```

- When you define a form in an HTML document, you can define the URL to be processed when the user activates the form, for example,

  ```
  <form action="/cgi-bin/cicsweb" method="POST">
  ```

  ECITEST uses this method to identify the script to process input from its forms.

7.3.1.2 Invoking GoServe Application Filter

GoServe uses the term *filter* for application scripts used to customize the way it works. You must write GoServe filters using REXX, though you can call programs written in other languages from your REXX filter if you wish.

When you are using servers that use the CGI interface, you tell the server to invoke an application script by specifying `/cgi-bin/` after the server name in the URL. GoServe, in contrast, calls a filter for every incoming Web browser request. GoServe's default filter is a REXX program named `gofilter.80`; This filter provides basic Web Server functions for retrieving documents and other objects in response to Web browser requests. You can modify this filter to perform any additional function you require.

You could add your entire application to `gofilter.80`. However it is probably a better idea to write your application as a separate filter, and make minimal modifications to `gofilter.80` so that it calls your filter. This is the approach that we took with ECITEST. You need to decide what is to trigger your application. For ECITEST, we decided to use an identifying string of `$ecitest` directly after the server name in the URL. That is, a URL starting with

```
http://baykal.sanjose.ibm.com/$ecitest/
```

triggers the ECITEST filter. To do this, we added the code shown in Figure 77 on page 130 to the default GoServe filter.

```
/*****************************************************************/
/* Modifications for CICS Web Interface.                         */
/*****************************************************************/
if translate(left(sel,8)) ='$ECITEST' then
  return ecitest(verb)                    /* call the ECI Tester */
```

Figure 77. Triggering ECITEST

7.3.2 Obtaining User Input from the Web Browser

After the Web server invokes your CGI script or GoServe filter, it needs to retrieve the information the user has entered into the form. The way you do this depends on the `METHOD` parameter you define on the HTML `<FORM>` tag. You have two options for METHOD: GET or POST. If you use GET, for example,

```
<FORM ACTION="/cgi-bin/ecitest" METHOD="GET">
```

then input data is appended to the URL string. This method is mainly provided for backward compatibility with earlier levels of HTML that did not support forms, and is limited in the amount of data that you can send. We highly recommended that you use the POST method instead, for example,

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

which sends the user's form input separately as part of the body of the input request.

Input data is returned as a string of variable names and values, with each name or value pair separated by an &, and an = between the variable name and the value. Blanks are sent as the + character, and some characters (including & = + and %) are encoded as their hex value, preceded by a % character. An example of an input string from the ECITEST login form is:

```
eci_userid=d%25flt&eci_password=&cicsweb_function=login&submit=Test+ECI
```

You read the form data from a CGI script like this:

```
if verb = 'POST' then do              /* Only support POSTs for forms */
  /* get the incoming data */
  eciparms = charin(,,value('CONTENT_LENGTH',,'OS2ENVIRONMENT'))
```

or from a GoServe filter as shown here:

```
if verb = 'POST' then do              /* Only support POSTs for forms */
  'read body var eciparms'                    /* get the incoming data */
```

You can then parse the string and assign the values to REXX variables. In ECITEST, we use a REXX stem variable to hold the input variables. For example, variable `eci_userid` is assigned to REXX variable `VAR.eci_userid` and so on. The following REXX procedure shown in Figure 78 on page 132 does this for us.

```
/*********************************************************************/
/* PARSEVAR: Extract variables from form (taken from MFC's GoRemote)  */
/*           e.g. variable eci_userid placed in var.eci_userid       */
/*********************************************************************/
parsevar: procedure expose var.
  /* set VAR.x variables from the incoming data stream */
  parse arg data
  VAR.=''                                 /* null string unless set  */
  data=data'&'                            /* set end condition       */
  do until data=''                        /* each value              */
    parse var data assign '&' data        /* split off name=value    */
    parse var assign name '=' value       /* separate                */
    value=translate(value, ' ', '2b090a0d'x) /* handle '+' tabs & CRLF*/
    name=translate(packur(name))          /* caseless name           */
    var.name=packur(value)                /* set, after URI decoding */
    /* say name 'is' value */
  end
  return ''
```

Figure 78. Extract Variables from Forms Using REXX

7.3.3 Maintaining Information about the State of Processing

When conducting a dialog with a Web browser user, all but the simplest of applications need to keep track of the state of the dialog so they know what function to perform. ECITEST is no exception–it needs to save the user's CICS user ID and password, make sure that an incoming request is from a user who has signed on, and know what function it is being asked to process.

When you develop an application that uses the Web to access CICS applications, you can keep state information in one or more of the Web browser, the Web server, and CICS itself. In ECITEST we use both the Web browser and the Web server (or more accurately the Web server CGI script or GoServe filter) to keep state information. ECITEST keeps the following items of information about the state of processing:

Session Handle When a user signs on, ECITEST generates a unique number to identify the session. Its value is kept with the Web browser in a hidden field in the form, for example:

```
<input type="hidden" NAME="handle" VALUE="123'">
```

ECITEST also uses the session handle to generate the name of a file used to store information about the state of processing on the Web server.

CICS User ID ECITEST keeps the CICS user ID both in the Web browser as a hidden field named eci_userid and in the state file on the Web server. When a request is received, ECITEST checks whether the two match. If they do not match, the state file is deleted from the Web server and the user is requested to reenter the user ID and password.

CICS Password ECITEST keeps the user's password in the state file on the Web server. ECITEST passes the user ID and password to CICS for validation on each call to the ECI.

Form Function ECITEST places a hidden field named cicsweb_function in each form. This field contains the function associated with that form, either `login` for the login form, or `Test ECI` for the main entry form. ECITEST uses this to determine what function you wish to perform.

Submit Function Each ECITEST form has one or more functions that you can perform by clicking on the Submit button. ECITEST uses this in conjunction with the Form Function variable to determine what function to perform.

7.3.4 Passing Data to and from CICS

ECITEST passes data to the CICS program using the CICS COMMAREA passed as part of the ECI call. When the CICS program ends, it uses the COMMAREA to transfer data back to ECITEST.

Using the ECI to call a CICS program, pass it a COMMAREA, and receive a COMMAREA in response is quite straightforward, especially using REXX and the RxCICS package. The code from ECITEST shown in Figure 79 shows what is required.

```
/* Initialize ECI call parameters */
eci.userid   = var.eci_userid
eci.password = var.eci_password
eci.program_name = var.eci_program_name
if var.eci_commarea = '' then var.eci_commarea = ' ' /* can't be 0 length
var.eci_commarea = ECI('ECI.', var.eci_commarea)
if eci.return_code = 0 then
  return BuildEciTest('ECITEST Form Processed.')
```

Figure 79. Initialize ECI Call Parameters

7.3.5 Generating Dynamic HTML Documents

If your application uses a document or form that is always the same, you can store the document on the Web server's disk and retrieve it and display it like any other HTML document. ECITEST uses this approach for the document that is used for you to enter your login details. This approach does not have the flexibility you need to do many of the things that you will want to do with your Web applications, however. Even the simplest of functions that ECITEST performs, such as building a dynamic message line, or displaying the contents of the returned CICS COMMAREA, cannot be achieved using static documents retrieved from disk. You need to generate dynamic HTML documents from your application to do this.

It is easy to create a dynamic HTML document and send it to a Web browser from your CGI script or GoServe filter. CGI scripts simply write the document to `stdout`. You can do this with the REXX Say command. You also need to build an HTTP header and send it to `stdout` before you send the dynamic document. You can do this yourself using the Say command, or use the `cgiutils` program included with the IBM Internet Connection Server and other Web servers. GoServe provides a number of ways to send dynamic documents to a Web browser, as well as facilities to assist you in building the HTTP response header.

7.3.6 Deleting Information about the State of Processing

ECITEST provides a Quit function that allows you to finish your ECITEST session. When you select this function, ECITEST deletes the file containing state data from the Web server.

In your own applications, you will probably want to implement a more sophisticated mechanism for cleaning up old state files. For example, you may elect to purge state files that have not been used for a period of time, and purge all state files at least once per day.

Chapter 8. A Simple CICS Access Program: CICSWEB

In this chapter we describe a sample application that shows how you can use the Web to access data stored in a CICS file

8.1 What Does CICSWEB Do?

CICSWEB allows CICS to function as the data source for GET requests from Web browsers. CICSWEB stores any type of object in a CICS VSAM file and retrieves it on request.

CICSWEB provides two functions:

- Object retrieval
- Administration

8.1.1 CICSWEB Object Retrieval Function

This function allows you to refer to any object stored by CICSWEB in a URL in the same way you reference an object stored on a conventional Web server. You can type this URL directly into your Web browser, or point to it with a hypertext link from an HTML document. If the object is a graphic, you can point to it from a document using an HTML <IMG> tag. If you store an HTML document in CICSWEB, you can use relative URLs to reference other objects stored in the same CICSWEB object server.

This function does not need to provide its own user interface, because it is accessed using standard Web browser facilities. The URL that you use to reference an object stored by CICSWEB is a little different depending on whether you are using the Domino Go Webserver or GoServe as your Web server. An example of a URL when using IBM Internet Connection Server is

```
http://bosporus.sanjose.ibm.com/cgi-bin/cicsweb/cicsweb.html
```

while for GoServe you need to use something like:

```
http://baykal.sanjose.ibm.com/$cicsweb/cicsweb.htm
```

(although with GoServe you can make your URL almost whatever you want).

Figure 80 on page 136 shows the major CICSWEB components.

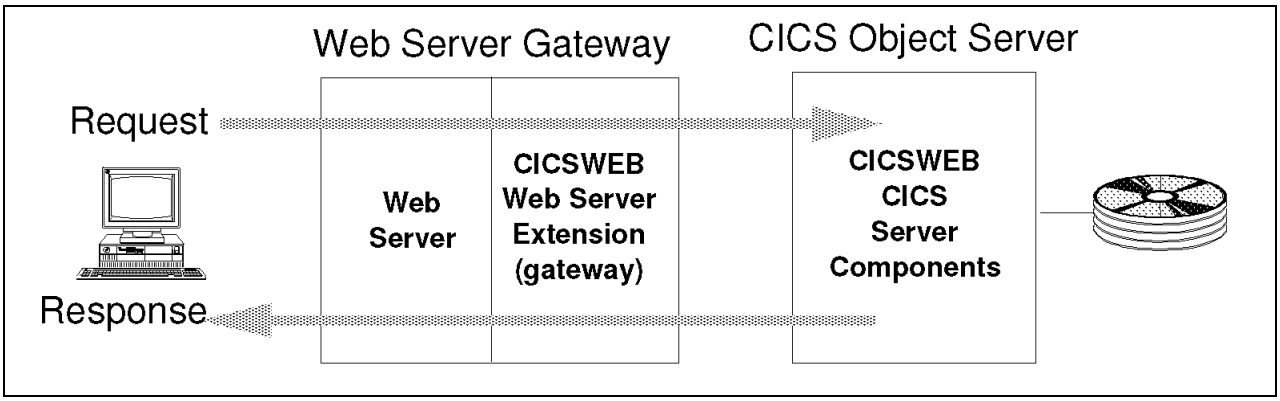


Figure 80. CICSWEB Major Components

8.1.2 CICSWEB Administration Function

To retrieve and display documents stored in CICS, we need to put them there first. CICSWEB provides three basic functions that allow you to manage the objects that it stores:

- Store an object
- Delete an object
- List objects

To use the CICSWEB administration function, enter your user ID and password on the CICSWEB Administration Login form, then click on the function you wish to perform.

Figure 81 on page 137 shows the CICSWEB Administration Login form.

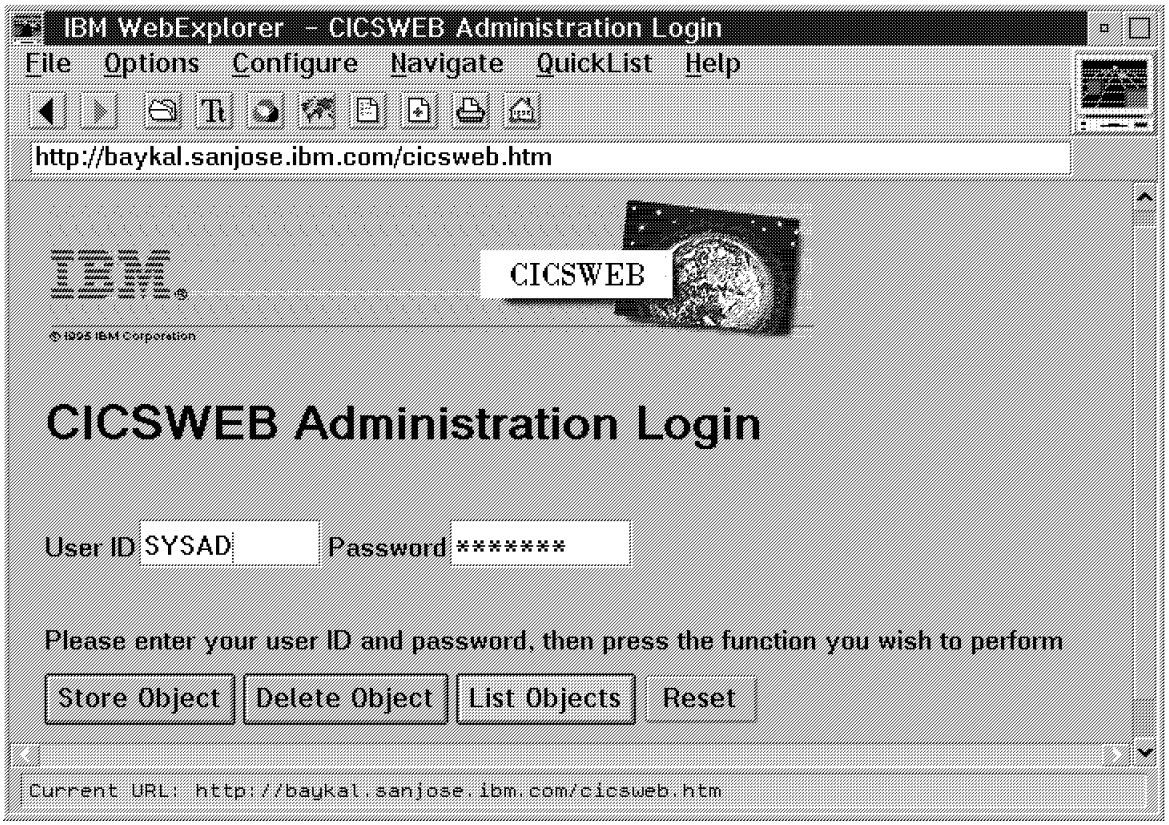


Figure 81. CICSWEB Administration Login Page

If you select the Store Object function, CICSWEB Administration displays the form shown in Figure 82 on page 138.

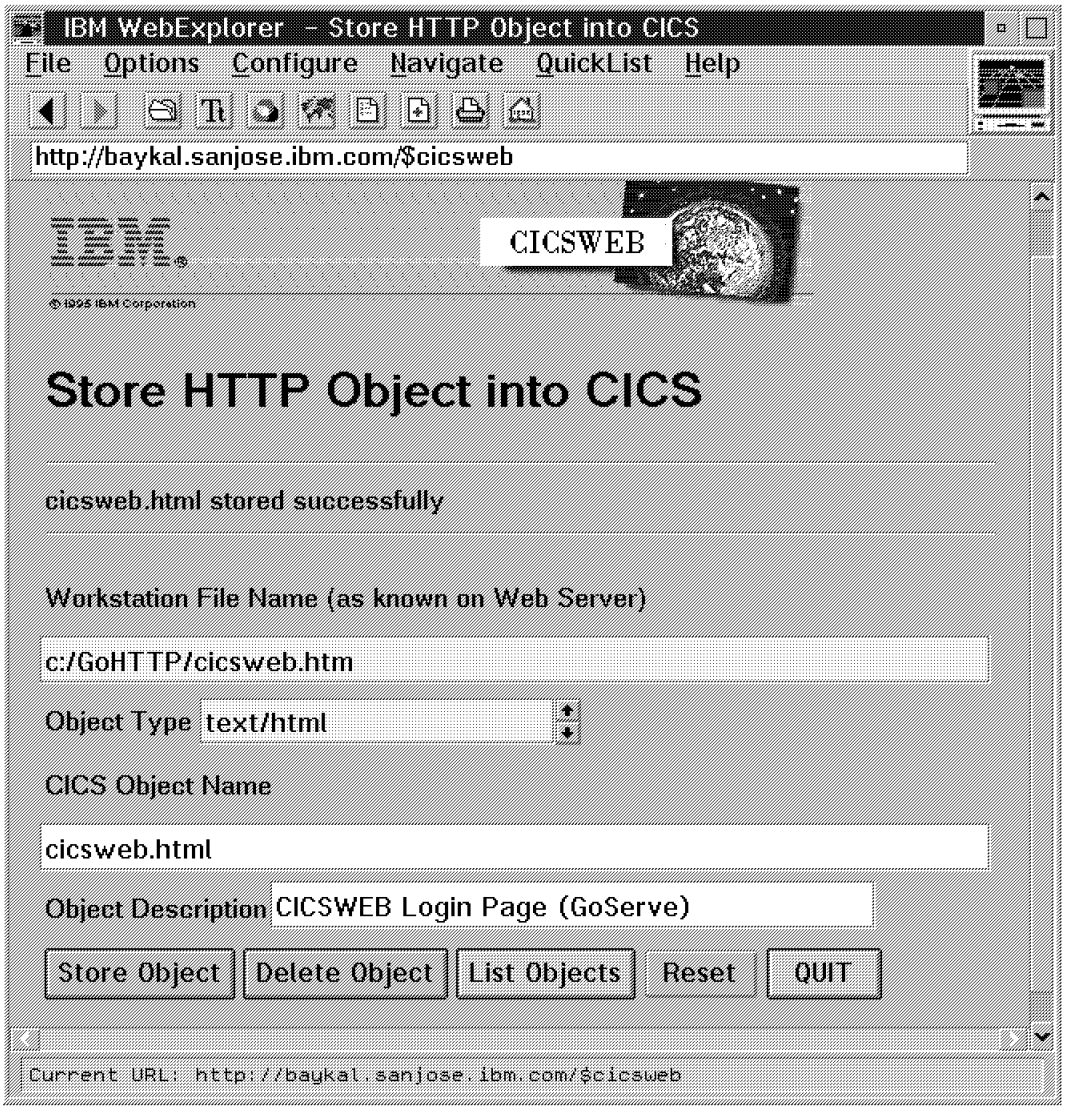


Figure 82. CICSWEB Store HTTP Object Page

Use this form to provide the following details:

- File name of the object you wish to store

  The file name you specify is the name that the Web server uses to access the file, since the Store Object function runs on the Web server.

- Object Type

  You select a value from the selection list provided, for example `text/html` for HTML documents, `text/plain` for simple ASCII text files, or `image/gif` for graphic interchange format (GIF) images. CICSWEB uses this value when it builds the HTTP header that identifies the type of object it is sending to a Web browser.

- CICS Object Name

  The name can be up to 48 characters, is case-sensitive, and should not include embedded spaces. The value is used to build the VSAM key used to store the object. While you can use any value that you like for the name, you

may find it convenient to use workstation-style hierarchical file names, for example `html/cicsweb.html` or `images/cwmast.gif`.

- Object Description

  You must specify a brief description of the object up to 30 characters long. CICSWEB displays this description when you use the List Objects function.

When you have entered the information required, click on the Store Object button (or press Enter) to complete the function. CICSWEB notifies you when the object has been successfully stored. If an object of the same name already exists, CICSWEB replaces it without warning; so be careful!

If you click on either the **Delete Object** or **List Objects** button instead, CICSWEB abandons the Store Object function and displays the form you requested.

If you select the Delete Object function, CICSWEB Administration displays the form shown in Figure 83 on page 139.

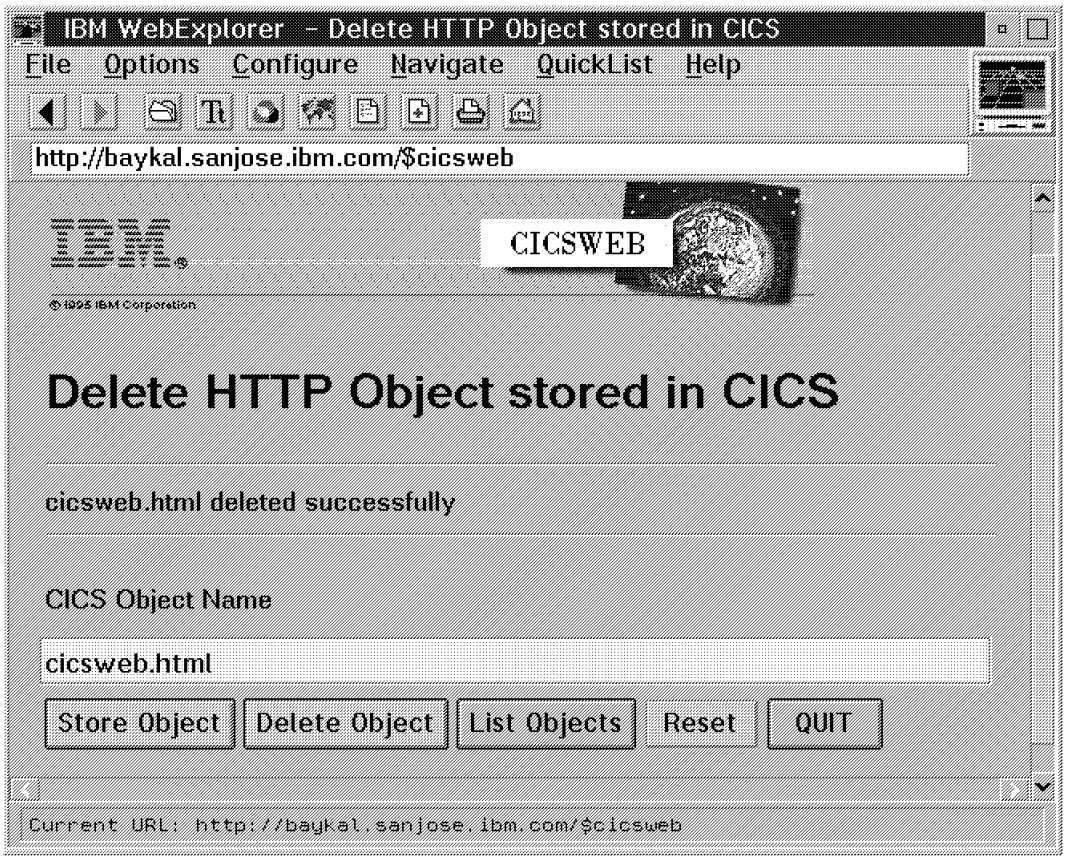


Figure 83. CICSWEB Delete HTTP Object Page

Type the name of the object exactly as you stored it, then click the **Delete Object** button (or press Enter) to complete the function. CICSWEB notifies you when the object has been successfully deleted.

If you click on either the **Store Object** or List **Objects button** instead, CICSWEB abandons the Delete Object function and displays the form you requested.

If you select the **List Objects** function, CICSWEB Administration displays the form shown in Figure 84 on page 140.

Figure 84. CICSWEB List HTTP Objects Page

Initially, the form is displayed without any objects listed. You can type a partial name (or generic key) for the objects you wish to display, then click on the **List Objects** button (or press Enter) to complete the function. Objects are displayed as shown in Figure 84 on page 140, in the EBCDIC collating sequence used by VSAM to store the objects. The list displays the object name, description, size, and content type. The object name is displayed as a hypertext link; so you can click on it to retrieve and display the object itself.

If you click on either the **Store Object** or **Delete Object** button instead, CICSWEB abandons the List Objects function and displays the form you requested.

8.2 CICSWEB Components and Interfaces

The CICSWEB application is built using components similar to those used by ECITEST, and described in 7.2, “ECITEST Components and Interfaces” on page 127:

- A Web server extension or gateway that provides CICSWEB application functions as well as the link between the Web environment and CICS
- CICS programs that perform CICSWEB application functions in the CICS environment.

You also need a Web browser to provide the user interface.

The major difference between CICSWEB and ECITEST is that while ECITEST provides a generic function to call any CICS program and display the data it returns, CICSWEB includes two CICS programs that perform specific functions in conjunction with the CICSWEB Web server component. These programs are CWSERVER, which stores, retrieves, and deletes objects managed by CICSWEB, and CWLIST, which performs the List Objects function.

8.3 CICSWEB Function Description

Like ECITEST, CICSWEB performs a number of functions required by most applications that use the Web to access CICS applications and resources. These include:

- Invoking an application-specific Web server extension or gateway
- Obtaining user input from the Web browser
- Maintaining information about the state of processing
- Passing data to and from CICS
- Generating dynamic HTML documents
- Deleting information about the state of processing on the Web server

CICSWEB performs these functions in the same way as ECITEST. See 7.3, “ECITEST Function Description” on page 129 for a description of how both ECITEST and CICSWEB perform these functions.

CICSWEB performs a number of functions over and above those performed by ECITEST, including:

- Adding data to CICS databases
- Using an extended logical unit of work (LUW)
- Retrieving data from CICS databases
- Minimizing network data traffic
- Using CICS user ID and password for validation
- Managing data conversion
- Generating HTML directly from a CICS application

We look at how CICSWEB performs each of these functions as an example of how you can perform similar functions in your own applications.

8.3.1 Adding Data to CICS Databases

The CICSWEB Web server CGI script or GoServe filter uses the CICS ECI to call a CICS program called CWSERVER. CICSWEB uses the CICS COMMAREA to transmit the contents of an object to CWSERVER. In addition to the object itself, the COMMAREA includes information such as the name of the object, its MIME type, length, description, and the name of the CICS file in which it is to be stored. CWSERVER builds a VSAM record key, and writes the object to the specified file.

Since multimedia objects such as images, sound, and video can be very large, CICSWEB breaks large objects into segments that are transmitted and stored individually. As implemented, CICSWEB allows for up to 999 segments of about 8000 bytes each.

8.3.2 Using an Extended Logical Unit of Work

CICSWEB can require multiple calls from the Web server environment to the CWSERVER program in CICS to transmit an object for storage. Because a partial object is not of much use, we need to ensure that either all or none of the object is stored. To do this, we need to define a logical unit of work (LUW) to CICS that spans as many ECI calls as necessary to transmit the entire object.

The ECI allows you to define the scope of the LUW using the `eci_extend_mode` parameter, which can be set to `ECI_EXTENDED` if the LUW is to be continued on subsequent calls, or `ECI_NO_EXTEND` for the last or only call in an LUW. Using the RxCICS interface, you specify this as follows:

```
/* Initialize ECI call parameters */
eci.userid   = var.eci_userid
eci.password = var.eci_password
eci.program_name = 'CWSERVER'
eci.extend_mode = "ECI_EXTENDED"           /* Multiple calls in LUW */

/* Call the program via ECI */
var.eci_commarea = ECI('ECI.', var.eci_commarea)
 :
 :
eci.extend_mode = "ECI_NO_EXTEND"       /* This is last call in LUW */
/* Call the program via ECI */
var.eci_commarea = ECI('ECI.', var.eci_commarea)
```

Alternatively, you can use `ECI_EXTENDED` for all your ECI calls to the CICS program, then terminate the LUW with an explicit call to commit changed resources as follows:

```
eci.program_name = ''
eci.extend_mode = "ECI_COMMIT"                          /* End LUW */
var.eci_commarea = ECI('ECI.', var.eci_commarea)
```

8.3.3 Retrieving Data from CICS Databases

CICSWEB retrieves an object from CICS by using the ECI to call CWSERVER to retrieve the first (or only) segment in an object. CWSERVER also returns the size of the object, the MIME type, and the number of segments. If the object spans more than one object, additional calls are made to CWSERVER to retrieve the remaining segments.

As the Web server component of CICSWEB receives each segment, it sends it immediately to the Web server. When using GoServe, each segment is sent to the Web browser immediately, so that retrieval from CICS and transmission to the Web browser are overlapped, improving response time.

8.3.4 Minimizing Network Data Traffic

When storing and retrieving objects, most data is transmitted in only one direction, either to or from CICS. The COMMAREA used to transmit the data remains the same size, however. The called program can neither shorten nor extend the COMMAREA.

Clearly, it would be detrimental to performance to transmit the largely empty COMMAREA from CICS back to the Web server after sending a segment of an object to CICS. To avoid this, you should make sure that the unused portion of the COMMAREA is padded with binary zeroes. Before transmitting a COMMAREA over a communications link, CICS scans the COMMAREA backwards from the end until the first nonzero character is detected. CICS transmits the COMMAREA only up to this character, and reconstitutes the full COMMAREA at the destination.

8.3.5 Using CICS User ID and Password for Validation

CICSWEB passes the CICS user ID and password to the ECI so that they can be validated on each call.

For CICSWEB administrative functions, the user ID and password are entered directly by the user on the login form, and maintained for the duration of the session on the Web server.

For retrieval requests, however, there is no way for the user to provide a user ID or password. The user could simply be clicking on a hypertext link in an HTML on display and have no idea that the document is stored in a CICS database. CICSWEB has a user ID and password embedded in the code that it uses when performing retrieve requests.

8.3.6 Generating HTML Directly from a CICS Application

As the number of documents that your application uses increases, so does the amount of code required to generate these documents.

CICSWEB generates three different dynamic HTML documents that you use to store, delete, and list documents managed by CICSWEB. To reduce the code required to generate these documents, CICSWEB segments each document into a header, body, and footer. The header and footer are kept the same for each document, so that common code can be used to generate them. Unique code is then required only to generate the body of each document.

The code to generate the document header and footer, as well as the document body for the Store Object and Delete Object functions is part of the Web server extension; the CICSWEB CGI script or GoServe filter. For the List Objects function, however, the HTML that displays the table of objects displayed is generated directly by the CWLIST CICS program, and simply incorporated into the document as it is being built on the Web server. See program CWLIST in

Appendix B, “CICSWEB Source Listings” on page 167 for the source code of the HTML-aware COBOL program that does this.

8.3.7 Managing Data Conversion

For our CICSWEB sample application, the CICS object server was a CICS/ESA system using EBCDIC character encoding, while the Web server uses the ASCII encoding of its OS/2 platform. Consequently, we need to ensure that ASCII-to-EBCDIC conversion is performed when required. When using CICS DPL (used by the CICS ECI), CICS can perform data conversion of the COMMAREA contents for you. Conversion is performed at the destination system if necessary, and in the case of a CICS/ESA system, defined for each program in the CICS DFHCNV table.

For CICSWEB, we have two CICS programs, CWSERVER and CWLIST, each with different data conversion needs. CWSERVER requires conversion of some control information in the COMMAREA, but not for the contents of any objects being transferred. CWLIST, on the other hand, deals only with character data, and its DFHCNV entry specifies that EBCDIC-ASCII translation be performed on the entire contents of the COMMAREA.

Chapter 9. CICS State Management Program: CICSSTAT

In this chapter we describe a sample application showing how you can use a generalized state management program to handle CICS resources related to the information about the state of processing, on behalf of CICS/ESA Web server programs that need to save data across invocations. To illustrate how to use state management, we also wrote a sample of a CICS Web server application that can build, modify, and destroy a list held on CICS temporary storage.

For full listings of the source code for these sample programs see Appendix C, “CICS/ESA State Management Sample” on page 193.

9.1 How Is CICSSTAT Invoked?

CICSSTAT can be invoked in three ways, and under a variety of transactions:

- The transaction CWBT causes CICSSTAT to be invoked directly. When invoked with this transaction, CICSSTAT performs its time out management function.
- The transaction CWBP causes CICSSTAT to release all information about the state of processing and to delete its anchor block from temporary storage.
- If it was not invoked under one of the above transactions, CICSSTAT assumes that it was linked to by a program wishing to use its state management facilities. In this case, it assumes that it was passed a COMMAREA containing information about the request.

9.2 What Does CICSSTAT Do?

CICSSTAT provides the following facilities to CICS/ESA Web pseudoconversation applications that need to store information about the state of processing across invocations of themselves:

- Allocates a unique identifier to a pseudoconversation.
- Allocates an area of CICS storage to be used to store information about the state of processing across invocations of the programs managing the pseudoconversation.
- Retrieves the information about the state of processing for a given pseudoconversation.
- Updates the information about the state of processing for a given pseudoconversation.
- Destroys the information about the state of processing for a given pseudoconversation.
- Destroys information about the state of processing for pseudoconversations that have not been active for longer than the specified time out period.
- Destroys information about the state of processing for all pseudoconversations.

9.2.1 CICSSTAT Single-Threading

We need a locking mechanism to be able to ensure that only one instance of CICSSTAT is executing at one moment. This avoids the possibility of concurrent updates to our state table, which could cause unpredictable results such as:

- Allocating the same unique identifier to two pseudoconversations
- Corrupting our state block chain.

We use the EXEC CICS ENQ and EXEC CICS DEQ commands to provide the locking. The ENQ defaults to SUSPEND, so if another task already has the lock, we wait until it is freed. Any excessive amount of waiting is likely to result in time outs on the Web browser or a CICS client if used. The lock is held by one task for a relatively short period, so this is a problem only when a very large number of Web transactions are being processed. If you have this problem, you may prefer to change the ENQ to have the NOSUSPEND option, so that control is returned immediately to CICSSTAT, which can then return an appropriate return code to the Web server.

9.2.2 The CICSSTAT Anchor Block

We need a piece of storage in which to put the CICSSTAT information about the state of processing. This area of storage must be accessible by multiple instances of CICSSTAT. We call this piece of storage the *state control block anchor* (SCBA). It resides on temporary storage; this means we need not worry about how to pass the address of the SCBA from one invocation of CICSSTAT to another. Any instance of CICSSTAT simply has to perform a READQ TS to copy the SCBA into its local storage. Our locking mechanism ensures that it cannot be updated between the time we read-in the TS record, and write back the new one. We put an eyecatcher at the front of the SCBA, so that it can be easily found in a dump. The SCBA contains the following information:

- Eyecatcher
- Forward pointer to first block of information about the state of processing
- Backward pointer to last block of information about the state of processing
- Unique identifier count
- Timestamp containing the time at which this SCBA was created

If you are implementing CICSSTAT as a Task Related User Exit (TRUE), then you can store the information in the Global Work Area for the TRUE, rather than on temporary storage.

If the READQ TS returns the QUEUEID error, then we know that this instance of CICSSTAT is the first since CICS was started, or since the last SCBA was deleted by an earlier CWBP transaction (see 9.3.2, “CICSSTAT Purge Processing” on page 149). In this situation, we build a new SCBA and write it to temporary storage. Any other error is treated as a logic error.

A message is written to the operator whenever an SCBA is created or deleted. You may wish to add extra messages to allow you to track any changes to your SCBA.

9.2.3 CICSSTAT COMMAREA Structure

If CICSSTAT is not running under the CWBP or CWBT transactions, it assumes that it was invoked by an EXEC CICS LINK issued by another program. In this case, the COMMAREA passed to CICSSTAT must contain the following information:

Eyecatcher The eyecatcher is provided for debugging purposes. If required, you can use the eyecatcher to identify the program that invoked CICSSTAT.

Function A 1-byte character field identifying the reason that CICSSTAT was invoked. This must be one of the following:

C (Create)

S (Store)

R (Retrieve)

D (Destroy)

Return Code A 1-byte return code indicating whether the request was successful, or why a request failed.

Handle The 4-byte unique identifier used to map information about the state of processing to a specific pseudoconversation. It is allocated to a specific pseudoconversation on the create call, and must be specified on all subsequent calls to CICSSTAT to update that state data.

9.2.4 Creating a State Block

When called with the create function, CICSSTAT creates a new state block, and adds it to its doubly linked list of state blocks. It uses an EXEC CICS GETMAIN specifying the SHARED parameter, so that the storage is not released at the termination of the task under which the GETMAIN is issued. Once the storage is acquired, pointers in the new and existing SCBs and the first and last state blocks pointers in the SCBA are updated to add the block to the chain.

If you decide you need variable-length information about the state of processing, you must change the GETMAIN to use a length passed in the COMMAREA, rather than a hard-coded value.

A 4-byte handle (an unsigned integer) that uniquely identifies this state block is returned to the caller. Any subsequent request to update this state block must provide this 4-byte handle.

The state block has the following structure:

STATE_EYECATCHER 4-byte eyecatcher for debugging purposes

STATE_FORWARD_PTR Pointer to the next information about the state of processing block in the chain

STATE_BACKWARD_PTR Pointer to the previous information about the state of processing block in the chain

STATE_HANDLE 4-byte field containing the unique identifier for this information about the state of processing

STATE_TIMESTAMP Packed decimal field containing a timestamp of the last time this information about the state of processing was accessed.

STATE_USER_DATA 256 bytes to store user data.

You are not tied to using 256 bytes for your information about the state of processing. If you need more or less, you can change the hard-coded values. If

the amounts of data vary greatly from one application to the next, you might prefer to change the COMMAREA so that the caller can specify a length of information about the state of processing required, and CICSSTAT can return a pointer to the state data, rather than a copy of it.

9.2.5 CICSSTAT Create Function

Application programs wishing to allocate some information about the state of processing use this function. The required input parameter is C in the function byte.

CICSSSTAT retrieves the unique identifier count from the anchor block, increments it, and uses the new value as the unique identifier for this state block. CICSSTAT also updates the counter.

A new area of storage for the new state block is obtained with GETMAIN, and chained off the list.

The timestamp field in the information about the state of processing block is updated with the current time.

The unique identifier is placed in the handle field in the COMMAREA to be passed back to the caller, and any information about the state of processing in the COMMAREA is copied into the state block user data area.

9.2.6 CICSSTAT Retrieve Function

Application programs wishing to retrieve the information about the state of processing associated with their pseudoconversation use this function. CICSSTAT expects two input parameters on the call:

The handle, which uniquely identifies the state block requested

The function byte, which must be set to R.

CICSSTAT goes through the chain of state blocks until it finds the specified unique identifier, and then copies the information about the state of processing in the state block into the COMMAREA, to be passed back to the application. The timestamp in the information about the state of processing is updated.

9.2.7 CICSSTAT Store Function

Application programs wishing to update the information about the state of processing associated with their pseudoconversation use this function. CICSSTAT expects these input parameters on the call:

- The handle, which uniquely identifies the state block requested
- The function byte,which must be set to S
- The user data to be put into the state block.

CICSSTAT goes through the chain of state blocks until it finds the specified unique identifier, and then copies the information about the state of processing in the COMMAREA into the state block. The timestamp in the information about the state of processing is updated.

9.2.8 CICSSTAT Destroy Function

Application programs wishing to destroy the information about the state of processing associated with their pseudoconversation use this function. CICSSTAT expects two input parameters on the call:

- The handle, which uniquely identifies the state block requested
- The function byte, which must be set to D.

CICSSTAT unchains the state block from the list, and then issues FREEMAIN to release the storage.

9.3 CICSTAT Routines

In this section we describe the routines CICSSTAT performs.

9.3.1 CICSSTAT Timeout Processing

When CICSSTAT is invoked using the transaction CWBT, it performs its timeout processing. The purpose is to destroy any state blocks that have expired; that is, the interval between the current time and the time at which those state blocks were last accessed is greater than the hard-coded timeout interval. You can change the value of the timeout interval to suit your installation.

Any application that attempts to access information about the state of processing that has been purged receives the no-match return code X'04' in the return code field in the COMMAREA following the link to CICSSTAT.

CWBT currently runs only if you enter the transaction at the terminal. You may prefer to automate this function by adding code to cause CWBT to reschedule itself to run at regular intervals. You can do this using the EXEC CICS START command with the interval parameter.

When invoked, the timeout code works through the list of state blocks, looking for those that have timed out. These blocks are unchained and released, and the calculations are done using times in absolute format. As a result, all the arithmetic is done in packed decimal format.

9.3.2 CICSSTAT Purge Processing

The purge routine is invoked by starting transaction CWBP. It can either be entered at a terminal or started by another program.

This is the routine invoked if we are running under transid CWBP. The routine starts at the beginning of the chain, and releases all the information about the state of processing blocks. When all have been released, it deletes our TS queue and issues a message to the console to say that state management has been terminated.

9.3.3 CICSSTAT Error Handling

The error handling in CICSSTAT is limited. If an error is detected, a message is written to the operator, and a bad return code is passed back to the caller in the COMMAREA. No distinction is made between catastrophic and non catastrophic errors; we always continue. You may prefer to change CICSSTAT so that it always terminates state management when there is an error, or you may prefer to

add some granularity, taking each error in turn, and making a decision about whether or not to continue state management.

9.4 Sample Scenario Using CICSSTAT

We coded a simple CICS/ESA program to illustrate how to use CICSSTAT state management for a CICS/ESA Web server application. It is a very simple application that allows you to build and display a list of items kept on CICS temporary storage. It uses CICSSTAT to hold state information about the name and address of the user, and the number of items on the temporary storage queue.

Figure 85 on page 150 shows the scenario we use.

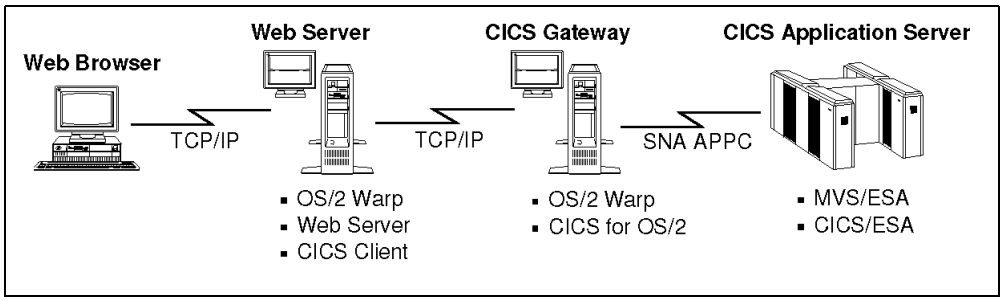


Figure 85. Sample Connectivity Scenario for CICSSTAT

We coded:

- A REXX CGI script (see Appendix C.1, “REXX CGI Script” on page 193)
- A COBOL CICS/ESA application (see Appendix C.2, “COBOL CICS/ESA Web Server Application Program” on page 195).

We placed most of the function in the CICS/ESA program rather than in the CGI script. All the state management and the forms building are done by our HTML-aware COBOL CICS/ESA application. We parsed the forms input in the CGI script rather than in the COBOL program, because it was quicker. This also reduces the amount of data that has to be passed to CICS/ESA in the COMMAREA. An alternative implementation of the CGI script could pass the unchanged forms input to the CICS/ESA application in the COMMAREA, for the CICS/ESA program to do the parsing.

After parsing the data, the CGI script invokes the ECI interface of the CICS client to invoke the CICS/ESA application.

The first time the ECI interface is invoked, there is no forms data for the CGI script to pass, so it simply passes a COMMAREA with the function field set to IDENTIFY. This tells the CICS/ESA application that this is a new request.

On subsequent invocations, the CGI script parses any forms data, which it passes to CICS/ESA in the COMMAREA.

All output from the CICS/ESA application is piped directly back to the Web browser, unchanged.

The COMMAREA passed to the COBOL application contains the following (but not on every call):

- Function to be performed
- Name of user (if known)
- Address of user (if known)
- Action requested by user
- Item to be added to the list

The function identifies which routine in the COBOL program is to be invoked:

Identify Call CICSSTAT with the create function to assign us a state block. Place the returned unique identifier in a form asking the user to register, update the function in the form to REGISTER, and return the form to the Web server.

Register Extract the unique identifier for this pseudoconversation from the form. Perform an EXEC CICS LINK to CICSSTAT with the retrieve function, passing the unique identifier, to make sure that we have a state data area.

Store the name and address in the information about the state of the processing area. Build a new form with function Add, putting name, address, and unique identifier in hidden fields, and send the form to the Web server.

Add Extract the unique identifier for this pseudoconversation from the form. Perform an EXEC CICS LINK to CICSSTATE passing the unique identifier, to make sure that we have information about the state of the processing area.

Check that the name and address in the form match those in the state data.

Check the action requested by the user (user can choose an **ADD** or a **DELETE** button):

If the user chooses **ADD**, do the WRITEQ and update the state data count, linking to CICSSTAT with function Store, then build a new form displaying the items currently in the list using a READQ TS to read each item from the temporary storage queue.

If the requested action is **DELETE**, issue a DELETEQ and perform an EXEC CICS LINK to CICSSTAT with the DESTROY function. Then build a form saying we have deleted the queue.

9.4.1 Error Handling

The error handling in this sample is limited. If an error is detected, we simply return an HTML document to the Web browser indicating what the problem was. It may be that the sample was invoked with an invalid function, an EXEC CICS call failed with an unexpected condition, or the information about the state of processing held in the form did not match that held in CICS/ESA.

9.4.2 Why Use Shared Storage Rather than Temporary Storage?

You may ask why this sample application uses CICS shared storage for the user information about the state of processing rather than CICS temporary storage. We chose this design primarily because of the better performance it offers, but other considerations were:

- It uses less file I/O
- It gives CICSSTAT a greater degree of control
- It makes it easier to convert to a task-related user exit

Chapter 10. CICS Sockets Sample

In this chapter we describe a simple application that illustrates how you might use the TCP/IP Sockets interface to access a CICS/ESA application from the Web. Although the problem might be solved much more easily using the new CWI, this sample illustrates very well the usage of the Sockets interface. For details on using the CICS to TCP/IP for MVS Sockets Interface, see *CICS/ESA and TCP/IP for MVS Sockets Interface*.

Source code for SOCKTEST components is included in Appendix D, “CICS/ESA Sockets Application Sample” on page 215.

10.1 SOCKTEST Environment

SOCKTEST consists of the following components:

- A CGI script that provides the link between the Web environment and CICS/ESA
- A set of C functions used by the CGI script to issue sockets calls
- A CIC/ESA application that performs the requested application function.

These components use the following interfaces:

- The mechanisms defined for the CICS Internet Gateway for the AIX Web server, which the Web server uses to communicate with the CGI script
- TCP/IP sockets, which the CGI script uses to communicate with CICS/ESA.

10.1.1 Connectivity Scenario

We used the IBM Internet Connection Server for AIX as our Web server.

We used TCP/IP for MVS Version 3.1, and the IBM CICS to TCP/IP for MVS Version 2 Release 2.1 Sockets Interface to allow the Web server to communicate with CICS/ESA Version 4.1 using the sockets API. Figure 86 on page 153 shows the major components required.

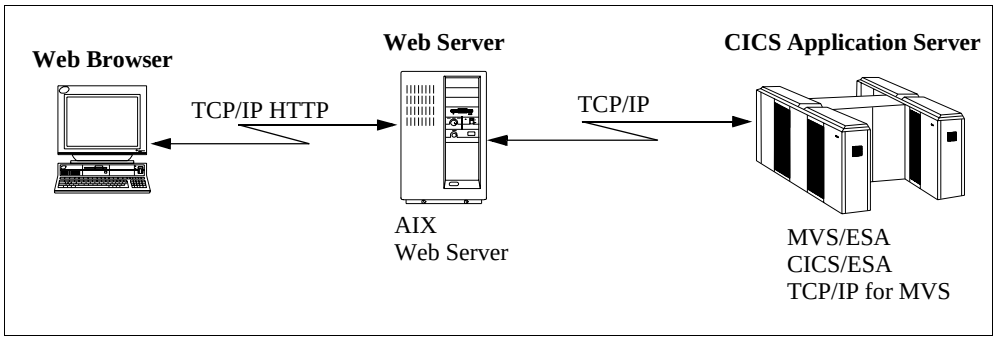


Figure 86. Sample Connectivity Scenario for SOCKTEST

10.2 What Does SOCKTEST Do?

SOCKTEST presents the user with a form asking for:

- The IP address of the MVS system on which CICS/ESA is running
- The port at which the sockets feature listener transaction is running (specified on CSKE)
- The name of the server transaction (TCP1).

After a pseudoconversation is established, the user is presented with a new form that allows the user to specify:

- Queue name
- Action to perform (READ,WRITE, or DELETE) on the queue
- Optional data for the queue. (The READ operation can only be performed on an existing queue).

If the queue does not exist, an error is returned to the Web browser

Function is split between the client and server components, with both being HTML-aware and handling information about the state of processing.

Note: The connections between the client and server are ephemeral. The connection is established, the data transferred, and the connection is then closed. This prevents excessive use of CICS resources as a result of holding connections during user think time. Because all client requests go through the supplied sockets feature listener transaction, we can have multiple requests to the same CICS region from many concurrent clients. The CGI script does the following:

1. Builds the initial form and sends it to the Web browser, with state data stored in hidden fields in the form.
2. Is reinvoked to process data input by the user.
3. Establishes a sockets connection with the CICS/ESA server program.
4. Sends the forms input to the CICS/ESA server program using sockets calls.
5. Retrieves response from the CICS/ESA server program using sockets calls.
6. Returns the HTML output by the CICS/ESA server program to the Web browser.
7. Goes back to Step 2.

The CICS/ESA server does the following:

1. Issues a sockets call to retrieve the forms input sent by the Web server.
2. Analyzes the user input to the form.
3. Performs the requested action on CICS temporary storage.
4. Builds a new HTML form to be returned to the Web browser.
5. Issues a sockets call to send the form to the Web server.

10.2.1 Generating Dynamic HTML Documents

The CICS/ESA application program generates HTML dynamically; this keeps the client code relatively simple, but means that there is a large working-storage section containing all the forms data.

Note that this application does not include a METHOD parameter for the form that is sent to the Web browser. In this case, the method defaults to GET. The forms data is therefore concatenated with the URL by the Web browser.

10.3 SOCKTEST Sample Programs Management

In this section, we explain how to build and run the sample programs.

10.3.1 Building the SOCKTEST Sample Programs

To build the client, issue the following commands on AIX:

```
cc -c sockets.c
cc -o client1.exe client1.c sockets.o
```

This builds an executable module called client1.exe, which should be copied into a directory from which it can be accessed by the Web server.

To build the CICS/ESA server program, modify the JCL provided in D.3, “MVS JCL to Compile COBOL Program” on page 228 and run the JCL on an MVS system that has TCP/IP and the CICS sockets feature installed. After the executable module is put in the load library, add and install a CICS system definition (CSD) file definition for the program (TCPSERV1) and the transaction (TCP1).

10.3.2 Running the SOCKTEST Sample Programs

Before running the client part of the sample, you must enable the sockets feature on the CICS/ESA system. Do this by running the CSKE transaction on CICS/ESA. After the sockets feature is enabled, you can run the client part by executing the client1.exe CGI script from a Web browser.

10.4 Information about the State of Processing for SOCKTEST

Hidden fields are used in the forms to allow information about the state of processing to be maintained for the pseudoconversation:

- The initial form, has three input fields: port, address, and transaction ID.
- Subsequent forms, have two input fields, queue name and queue data, and a selection box for the operation.
- The client knows about all six variables.
- In the first form only the port, address, and transaction ID are displayed; the other three variables are hidden.
- In subsequent forms, the port, address, and transaction ID are hidden but the other three variables are displayed.
- The hidden port, address, and transaction ID variables are required because the client basically makes a new call to CICS for each operation. The client must therefore have this information each time it connects to the CICS region using socket calls.

Appendix A. ECITEST Source Listings

This Appendix includes source listings for the various ECITEST components:

- Login HTML document for use with the IBM Internet Connection Server.
- Login HTML document for use with GoServe.
- REXX CGI script for use with the IBM Internet Connection Server.
- REXX filter for use with GoServe.

A.1 ECITEST.HTML: Login HTML Document for Use with the IBM Internet Connection Server

```
<!doctype html public "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>Test CICS ECI Program</title></head>
<body>
<img src="icons/cwmast.gif" align=center><p>
<h1>Test CICS ECI Program</h1>
<form action="/cgi-bin/ecitest" method="POST">
Enter CICS Login:
User ID <input NAME="eci_userid" VALUE="" SIZE=8>
Password <input Type="password" NAME="eci_password" VALUE="" SIZE=8> <P>
<input type="hidden" NAME="cicsweb_function" VALUE=login><P>
<br>
Please enter your user ID and password,
then click <strong>Test ECI</strong> to proceed<p>
<input type="submit" name="submit" value="Test ECI">
<input type="reset">
</form>
</body></html>
```

A.2 ECITEST.HTM: Login HTML Document for Use with GoServe

```
<!doctype html public "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>Test CICS ECI Program</title></head>
<body>
<img src="cicsweb/cwmast.gif" align=center><p>
<h1>Test CICS ECI Program</h1>
<form action="$ecitest" method="POST">
Enter CICS Login:
User ID <input NAME="eci_userid" VALUE="" SIZE=8>
Password <input Type="password" NAME="eci_password" VALUE="" SIZE=8> <P>
<input type="hidden" NAME="cicsweb_function" VALUE=login><P>
<br>
Please enter your user ID and password,
then click <strong>Test ECI</strong> to proceed<p>
<input type="submit" name="submit" value="Test ECI">
<input type="reset">
</form>
</body></html>
```

A.3 ECITEST.CMD: REXX CGI Script for Use with the IBM Internet Connection Server

```
/*********************************************************************/
/* CGI-BIN script to invoke sample CICS-ECI application tester       */
/*   - Saves user ID and password to file on Web Server              */
/*   - Generates and displays form with program name and COMMAREA    */
/*   - Calls the program, and displays COMMAREA returned             */
/*                                                                   */
/* Invokes RxECI to call specified application                       */
/*                                                                   */
/*********************************************************************/
verb = value('REQUEST_METHOD',,'OS2ENVIRONMENT')
/* '@CGIUTILS -ct text/html'   */
if verb = 'POST' then do             /* Only support POSTs for forms */
```

```
  /* get the incoming data */
  eciparms = charin(,,value('CONTENT_LENGTH',,'OS2ENVIRONMENT'))
  /* Parse input vars from form into var.x  */
  call parsevar eciparms
end
else                      /* not a form, return error message to user */
  return response('badreq','is not a valid REQUEST_METHOD: 'verb)

/* Check if new login attempt */
if translate(var.cicsweb_function) = 'LOGIN' then do   /* new session */
  state = startsession()
  if state \= '' then           /* display any error message returned */
    return response('badreq', state)
end /* LOGIN */
else do                 /* Check that we have a valid current session */
  session = checksession()
  if session \= '' then         /* display any error message returned */
    return response('badreq', session)
end /* check session status */

/* Check if processing an incoming form, or a request for a new form  */
/* - each form has a hidden field (cicsweb_function) with a value that*/
/*   can be checked against the SUBMIT function the user has selected */
if var.submit = '' then                     /* User has pressed ENTER */
  var.submit = var.cicsweb_function
if var.submit \= var.cicsweb_function then do     /* request new form */
  /* Now build the document for requested function and send to user   */
  select
    when var.submit = "Test ECI" then
      return BuildEciTest('')
    when var.submit = "QUIT" then do
      return QuitWeb(var.handle)
      end
    otherwise return response('badreq', 'asked for unknown form')
  end /* select form to display */
end /* build new form */
else do                                   /* Process input from form */
  /* Now build the document for requested function and send to user   */
  select
    when var.submit = "Test ECI" then
      return ProcessEciTest()
    otherwise return response('badreq', 'asked for unknown form')
  end /* Process Form Input */
end /* build new form */

return say 'We should never get here'

/*********************************************************************/
/* BuildEciTest: Generic form to execute pgm via ECI                 */
/*               COMMAREA returned from program is displayed         */
/*********************************************************************/
BuildEciTest: procedure expose var. eci.
  parse arg message
  title    = 'Test ECI Program'
  function = 'Test ECI'
  if message = '' then                   /* generate default message */
    message  = 'Please enter information and click button for function to perform'
  if length(var.eci_commarea) < 60 then do /* generate default message*/
    var.eci_commarea = var.eci_commarea||copies('.', 60-length(var.eci_commarea))
    eci.commarea_length = 60
  end
  if var.eci_program_name = '' then var.eci_program_name = 'SAMPLE'
  else var.eci_program_name = translate(var.eci_program_name) /* upper case */
  call Build_header title, function, var.handle, var.eci_userid, message
  say 'Program Name <input NAME="eci_program_name" SIZE=8 value="'
  say var.eci_program_name||'"><p>'
  say 'Commarea<p><pre>'
  if datatype(eci.commarea_length, 'N') then do
    say '|...+....1....+....2....+....3....+....4....+....5....+....6....<p>'
    do i = 1 to (eci.commarea_length % 64) + 1
      say substr(var.eci_commarea, 1 + 64*(i -1), 64)||'<p>'
    end
  end
  say '<textarea NAME="eci_commarea" rows=5 cols=64>'
  say var.eci_commarea'</textarea><p></pre>'
  call Build_footer
  return ''
```

```
/*********************************************************************/
/* Build_Header: Build document and form header for html document    */
/*********************************************************************/
Build_Header: procedure
  parse arg title, function, handle, userid, message
  /* Call CGIUTILS to build HTML Header */
  '@cgiutils -ct text/html'
  say '<!doctype html public "-//IETF//DTD HTML 2.0//EN">'
  say '<html><head>'
  say '<title>'title'</title></head>'
  say '<body>'
  say '<h1>'title'</h1>'
  say '<hr>'message'<hr>'
  say '<form action="/cgi-bin/ecitest" method="POST">'
  say '<input type="hidden" NAME="cicsweb_function" VALUE="'||function||'">'
  say '<input type="hidden" NAME="handle" VALUE="'||handle||'">'
  say '<input type="hidden" NAME="eci_userid" VALUE="'||userid||'">'
  return ''

/*********************************************************************/
/* Build_Footer: Build document and form footer for html document    */
/*********************************************************************/
Build_Footer: procedure
  say '<input type="submit" name="submit" value="Test ECI">'
  say '<input type="reset">'
  say '<input type="submit" name="submit" value="QUIT">'
  say '</form></body></html>'
  return ''

/*********************************************************************/
/* ECIMSGTEXT: Display Error Message from ECI Call                   */
/*           e.g. variable eci_userid placed in var.eci_userid       */
/*********************************************************************/
ecimsgtext: procedure  expose ECI.
  msgfile = "rxeci.msg"
  msgtext = 'Unknown ECI return code'
  do until lines(msgfile) = 0
    line = linein(msgfile)
    parse var line returncode message
    if ECI.return_code = returncode then msgtext = message
  end
  rc = lineout(msgfile) /* close file */
  return msgtext

/*********************************************************************/
/* PARSEVAR: Extract variables from form                             */
/*           e.g. variable eci_userid placed in var.eci_userid       */
/*********************************************************************/
parsevar: procedure expose var.
  /* set VAR.x variables from the incoming data stream */
  parse arg data
  VAR.=''                                  /* null string unless set */
  data=data'&'                             /* set end condition */
  do until data=''                         /* each value */
    parse var data assign '&' data         /* split off name=value */
    parse var assign name '=' value        /* separate */
    value=translate(value, ' ', '2b090a0d'x) /* handle '+', tabs, and CRLF */
    name=translate(packur(name))           /* caseless name */
    var.name=packur(value)                 /* set, after URI decoding */
    /* say name 'is' value */
  end
  return ''

/*********************************************************************/
/* ProcessEciTest: Process Form to Exercise ECI program              */
/*********************************************************************/
ProcessEciTest: procedure expose var. eci.
  /* register RxECI functions */
  rc = rxfuncdrop('ECI')
  If rxfuncquery('ECI') Then Do
     If rxfuncadd('ECI', 'RXECI', 'RxECI' ) Then
       return BuildEciTest('Error initializing RxECI')
     say "ECI registered"
  End

  /* Initialize ECI call parameters */
  eci.userid   = var.eci_userid
  eci.password = var.eci_password
```

```
  eci.program_name = var.eci_program_name
  if var.eci_commarea = '' then var.eci_commarea = ' ' /* can't be 0 length */
  var.eci_commarea = ECI('ECI.', var.eci_commarea)
  if eci.return_code = 0 then
    return BuildEciTest('ECITEST Form Processed.')
  return BuildEciTest('ECI Error. RC ='eci.return_code EciMsgText() ECI.abend_code)

/*********************************************************************/
/* QuitWeb: Delete State File and Return to Login Screen             */
/*********************************************************************/
QuitWeb: procedure
  parse arg handle
  port = extract('serverport')
  '@del tmp\CW'||handle||'.'||port
  '@cgiutils -ct text/html -status 302 -uri ecitest.html -noel'
  if port = 80 then port = ''
  else port = ':'||port
  say 'Location: http:///'extract(serveraddr)||port'/ecitest.html'
  say ''
  return ''

/*********************************************************************/
/* STARTSESSION: build server session-state file                     */
/*********************************************************************/
startsession: procedure expose var. eci.
  if (var.eci_userid = '') | (var.eci_password = '') then
    return 'needs to include user ID and password'
  var.handle = extract(transaction)           /* Save for later use   */
  /* generate new state file name */
  statefile = 'tmp/CW'||var.handle'.'extract(serverport)
  call lineout statefile, var.eci_userid, 1   /* userid to state file */
  call lineout statefile, var.eci_password    /* pw to state file     */
  return ''

/*********************************************************************/
/* CHECKSESSION: validate request against existing state file        */
/*   Checks that userid in state file matches userid sent from form  */
/*********************************************************************/
checksession: procedure expose var. eci.
  /* generate state file name */
  statefile = 'tmp/CW'||var.handle'.'extract(serverport)
  if stream(statefile, 'c', 'query exists') = '' then    /* Not found */
    return 'No current session exists.  Please login'
  if linein(statefile, 1) \= var.eci_userid then
    return "We don't have a record of your user id.  Please login again"
  var.eci_password = linein(statefile)      /* get pw from state file */
  rc = stream(statefile, 'c', 'close')      /* Close the state file   */
  return ''

/*********************************************************************/
/* The following functions are designed to be compatible with GoServe */
/* functions, and have been written to facilitate porting GoServe    */
/* filters to use the cgi-bin interface.                             */
/*********************************************************************/

/*********************************************************************/
/* Clientname: Return server name (modelled after GoServe)           */
/*********************************************************************/
Clientname: procedure
  return value('REMOTE_HOST',,'OS2ENVIRONMENT')

/*********************************************************************/
/* Extract: Return value of various settings                         */
/*********************************************************************/
extract: procedure
  parse upper arg var
  select
    when var = "DATADIR" then
      return value('REMOTE_ADDR',,'OS2ENVIRONMENT')
    when var = "CLIENTADDR" then
      return value('REMOTE_ADDR',,'OS2ENVIRONMENT')
    when var = "SERVERADDR" then
      return value('SERVER_NAME',,'OS2ENVIRONMENT')
    when var = "SERVERPORT" then
      return value('SERVER_PORT',,'OS2ENVIRONMENT')
    when var = "SERVERPROTOCOL" then
      return value('SERVER_PROTOCOL',,'OS2ENVIRONMENT')
    when var = "SERVERSOFTWARE" then
```

```
      return value('SERVER_SOFTWARE',,'OS2ENVIRONMENT')
    when var = "TRANSACTION" then do          /* get a session handle */
      /* return a quasi-unique number for building a session handle   */
      statefile = 'cwhandle.seq'
      handle = 1
      if stream(statefile, 'c', 'query exists') = '' then /* Not found*/
        call lineout statefile, '1', 1             /* Initialise file */
      else do                     /* Increment handle sequence number */
        handle = linein(statefile, 1) + 1
        call lineout statefile, handle, 1              /* Update file */
      end
      rc = stream(statefile, 'c', 'close')     /* Close the seq. file */
      return handle
    end
    otherwise return 'UNKNOWN'
  end

/*********************************************************************/
/* Packur:                                                           */
/* This procedure takes an input string, and converts characters     */
/* encoded as escape sequences (e.g. %xx) back to the character that */
/* they represent.                                                   */
/*********************************************************************/
packur: procedure
  parse arg outstring '%' rest
  do while length(rest) > 1
    outstring = outstring||x2c(substr(rest,1,2))      /* decode value */
    rest = substr(rest,3)
    parse var rest next '%' rest
    outstring = outstring||next
  end
  return outstring

/*********************************************************************/
/* Response: Send an error response to client                        */
/*********************************************************************/
response: procedure
  parse arg request, message
  select
    when request='badreq'   then use='400 Bad Request Syntax'
    when request='notfound' then use='404 Not found'
    when request='forbid'   then use='403 Forbidden'
    when request='unauth'   then use='401 Unauthorized'
  end  /* Add others to this list as needed */
  /* Now set the response and build the response file */
  parse var use code text
  '@cgiutils -ct text/html -status ' code
  say '<!doctype html public "-//IETF//DTD HTML 2.0//EN">'
  say "<html><head><title>"text"</title></head>"
  say "<body><h2>Sorry...</h2>"
  say "<p>The request from your Web client" message"."
  say "<hr><em>HTTP response code:</em>" code '['text']'
  say "<br><em>From server at:</em>" servername()
  say "<br><em>Running:</em>" server()
  say "</body></html>"
  return ''                                  /* [called as function] */


/*********************************************************************/
/* Servername: Return server name (modelled after GoServe)           */
/*********************************************************************/
Servername: procedure
  return value('SERVER_NAME',,'OS2ENVIRONMENT')

/*********************************************************************/
/* Server: Return server software (modelled after GoServe)           */
/*********************************************************************/
Server: procedure
  return value('SERVER_SOFTWARE',,'OS2ENVIRONMENT')
```

A.4 ECITEST.80: REXX Filter for Use with GoServe

```
/*********************************************************************/
/* GoServe filter to invoke sample CICS-ECI application tester       */
/* Invokes RxECI to call specified application                       */
/*                                                                   */
/* Arguments:                                                        */
/*   verb  - how called (only POST supported)                        */
/*   pgmid - selector after cgi-bin/cicsweb/ stripped from front     */
/*           (optional - used to provide program name to test)       */
/*********************************************************************/
parse arg verb
if verb = 'POST' then do              /* Only support POSTs for forms */
  'read body var eciparms'                    /* get the incoming data */
  if rc=-4 then                                    /* body too large */
    return response('badreq', 'sent too much data')
  if rc<>0 then                           /* e.g., invalid HTTP header */
    return response('badreq', 'sent data that could not be read')
  /* Parse input vars from form into var.x */
  call parsevar(eciparms)
end
else                      /* not a form, return error message to user */
  return response('badreq','is not a valid request')

/* Check if new login attempt */
if translate(var.cicsweb_function) = 'LOGIN' then do   /* new session */
  state = startsession()
  if state \= '' then           /* display any error message returned */
    return response('badreq', state)
end /* LOGIN */
else do                  /* Check that we have a valid current session */
  session = checksession()
  if session \= '' then         /* display any error message returned */
    return response('badreq', session)
end /* check session status */

/* Check if processing an incoming form, or a request for a new form  */
/* - each form has a hidden field (cicsweb_function) with a value that*/
/*   can be checked against the SUBMIT function the user has selected */
if var.submit \= var.cicsweb_function then do     /* request new form */
  /* Now build the document for requested function and send to user   */
  select
    when var.submit = "Test ECI" then
      return BuildEciTest('')
    when var.submit = "QUIT" then do
      return QuitWeb(var.handle)
      end
    otherwise return response('badreq', 'asked for unknown form')
  end /* select form to display */
end /* build new form */
else do                                  /* Process input from form */
  /* Now build the document for requested function and send to user   */
  select
    when var.submit = "Test ECI" then
      return ProcessEciTest()
    otherwise return response('badreq', 'asked for unknown form')
  end /* Process Form Input */
end /* build new form */

return say 'We should never get here'

/*********************************************************************/
/* BuildEciTest: Generic form to execute pgm via ECI                 */
/*               COMMAREA returned from program is displayed         */
/*********************************************************************/
BuildEciTest: procedure expose var. eci.
parse arg message
title    = 'Test ECI Program'
function = 'Test ECI'
if message = '' then                     /* generate default message */
  message  = 'Please enter information and click button for function to perform'
if length(var.eci_commarea) < 60 then do  /* generate default message */
  var.eci_commarea = var.eci_commarea||copies('.', 60-length(var.eci_commarea))
  eci.commarea_length = 60
end
if var.eci_program_name = '' then var.eci_program_name = 'SAMPLE'
else var.eci_program_name = translate(var.eci_program_name) /* upper case */
```

```
crlf='0d0a'x
html = ''
html = Build_header(html, title, function, var.handle, var.eci_userid, message)
html = html||'Program Name <input NAME="eci_program_name" SIZE=8 value="'
html = html||var.eci_program_name||'"><p>'crlf
html = html||'Commarea<p><pre>'crlf
if datatype(eci.commarea_length, 'N') then do
  html = html||'|...+....1....+....2....+....3....+....4....+....5....+....6....<p>'
  do i = 1 to (eci.commarea_length % 64) + 1
    html = html||substr(var.eci_commarea, 1 + 64*(i -1), 64)||'<p>'crlf
  end
end
html = html||'<textarea NAME="eci_commarea" rows=5 cols=64>'
html = html||var.eci_commarea'</textarea><p></pre>'crlf
html = Build_footer(html)
'VAR TYPE text/html AS ECITest NAME html'
return ''

/*********************************************************************/
/* Build_Header: Build document and form header for html document    */
/*********************************************************************/
Build_Header: procedure
parse arg html, title, function, handle, userid, message
crlf='0d0a'x
html = html||'<!doctype html public "-//IETF//DTD HTML 2.0//EN">'crlf
html = html||'<html><head>'crlf
html = html||'<title>'title'</title></head>'crlf
html = html||'<body>'crlf
html = html||'<h1>'title'</h1>'crlf
html = html||'<hr>'message'<hr>'crlf
html = html||'<form action="$ecitest" method="POST">'crlf
html = html||'<input type="hidden" NAME="cicsweb_function" VALUE="'||function||'">'crlf
html = html||'<input type="hidden" NAME="handle" VALUE="'||handle||'">'crlf
html = html||'<input type="hidden" NAME="eci_userid" VALUE="'||userid||'">'crlf
return html

/*********************************************************************/
/* Build_Footer: Build document and form footer for html document    */
/*********************************************************************/
Build_Footer: procedure
parse arg html
crlf='0d0a'x
html = html||'<input type="submit" name="submit" value="Test ECI">'crlf
html = html||'<input type="reset">'crlf
html = html||'<input type="submit" name="submit" value="QUIT">'crlf
html = html||'</form></body></html>'crlf
return html

/*********************************************************************/
/* ECIMSGTEXT: Display Error Message from ECI Call                   */
/*           e.g. variable eci_userid placed in var.eci_userid       */
/*********************************************************************/
ecimsgtext: procedure  expose ECI.
  msgfile = "rxeci.msg"
  msgtext = 'Unknown ECI return code'
  do until lines(msgfile) = 0
    line = linein(msgfile)
    parse var line returncode message
    if ECI.return_code = returncode then msgtext = message
  end
  rc = lineout(msgfile) /* close file */
return msgtext

/* return code from ECI */
  ECI.return_code       = ""                     /* respone  */

/*********************************************************************/
/* PARSEVAR: Extract variables from form                             */
/*           e.g. variable eci_userid placed in var.eci_userid       */
/*********************************************************************/
parsevar: procedure expose var.
  /* set VAR.x variables from the incoming data stream */
  parse arg data
  VAR.=''                                  /* null string unless set */
  data=data'&'                             /* set end condition */
  do until data=''                         /* each value */
    parse var data assign '&' data         /* split off name=value */
    parse var assign name '=' value        /* separate */
```

```
    value=translate(value, ' ', '2b090a0d'x) /* handle '+', tabs, and CRLF */
    name=translate(packur(name))             /* caseless name */
    var.name=packur(value)                   /* set, after URI decoding */
    /* say name 'is' value */
  end
  return ''

/*********************************************************************/
/* ProcessEciTest: Process Form to Exercise ECI program              */
/*********************************************************************/
ProcessEciTest: procedure expose var. eci.
  /* register RxECI functions */
  rc = rxfuncdrop('ECI')
  If rxfuncquery('ECI') Then Do
     If rxfuncadd('ECI', 'RXECI', 'RxECI' ) Then
       return BuildEciTest('Error initializing RxECI')
     say "ECI registered"
  End

  /* Initialize ECI call parameters */
  eci.userid   = var.eci_userid
  eci.password = var.eci_password
  eci.program_name = var.eci_program_name
  if var.eci_commarea = '' then var.eci_commarea = ' ' /* can't be 0 length */
  var.eci_commarea = ECI('ECI.', var.eci_commarea)
  if eci.return_code = 0 then
    return BuildEciTest('ECITEST Form Processed.')
  return BuildEciTest('ECI Error. RC ='eci.return_code EciMsgText() ECI.abend_code)

/*********************************************************************/
/* QuitWeb: Delete State File and Return to Login Screen             */
/*********************************************************************/
QuitWeb: procedure
parse arg handle
address cmd
'del tmp\CW'||handle||'.'||extract(serverport)
return 'FILE NAME '||extract(datadir)||'ecitest.htm'

/*********************************************************************/
/* RESPONSE: Standard (mostly error) responses (from GoRemote)       */
/*                                                                   */
/* Arguments are: response type and extended message information.     */
/* It uses the VAR command to send message document to Web Browser   */
/*********************************************************************/
response: procedure
  parse arg request, message
  select
    when request='badreq'   then use='400 Bad Request Syntax'
    when request='notfound' then use='404 Not found'
    when request='forbid'   then use='403 Forbidden'
    when request='unauth'   then use='401 Unauthorized'
  end  /* Add others to this list as needed */
  /* Now set the response and build the response file */
  'RESPONSE HTTP/1.0' use     /* Set HTTP response line */
  parse var use code text
  crlf='0d0a'x; out=''
  out=out||'<!doctype html public "-//IETF//DTD HTML 2.0//EN">'crlf
  out=out||"<html><head><title>"text"</title></head>"crlf
  out=out||"<body><h2>Sorry...</h2>"crlf
  out=out||"<p>The request from your Web client" message"."crlf
  out=out||"<hr><em>HTTP response code:</em>" code '['text']'crlf
  out=out||"<br><em>From server at:</em>" servername()crlf
  out=out||"<br><em>Running:</em>" server()crlf
  out=out||"</body></html>"crlf
  'VAR TYPE text/html AS Response NAME out'  /* send it */
  return ''                                 /* [called as function] */

/*********************************************************************/
/* STARTSESSION: build server session-state file                     */
/*********************************************************************/
startsession: procedure expose var. eci.
  if (var.eci_userid = '') | (var.eci_password = '') then
    return 'needs to include user ID and password'
  var.handle = extract(transaction)          /* Save for later use   */
  /* generate new state file name */
  statefile = 'tmp/CW'||var.handle'.'extract(serverport)
  call lineout statefile, var.eci_userid, 1   /* userid to state file */
  call lineout statefile, var.eci_password    /* pw to state file     */
```

```
  return ''

/*********************************************************************/
/* CHECKSESSION: validate request against existing state file        */
/*   Checks that userid in state file matches userid sent from form  */
/*********************************************************************/
checksession: procedure expose var. eci.
  /* generate state file name */
  statefile = 'tmp/CW'||var.handle'.'extract(serverport)
  if stream(statefile, 'c', 'query exists') = '' then    /* Not found */
    return 'No current session exists.  Please login'
  if linein(statefile, 1) \= var.eci_userid then
    return "We don't have a record of your user id.  Please login again"
  var.eci_password = linein(statefile)      /* get pw from state file */
  rc = stream(statefile, 'c', 'close')      /* Close the state file   */
  return ''
```

Appendix B. CICSWEB Source Listings

This Appendix includes source listings for the various CICSWEB components:

- Login HTML document for use with the IBM Internet Connection Server.
- Login HTML document for use with GoServe.
- REXX CGI script for use with the IBM Internet Connection Server.
- REXX filter for use with GoServe.
- CICS COBOL program to store/retrieve and delete objects.
- CICS COBOL program to list objects.
- CICS data conversion table.
- VSAM file definition.

B.1 CICSWEB.HTML: Login HTML Document for Use with the Domino Go Webserver

```
<!doctype html public "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>CICSWEB Administration Login</title></head>
<body>
<img src="icons/cwmast.gif" align=center><p>
<h1>CICSWEB Administration Login</h1>
<form action="/cgi-bin/cicsweb" method="POST">
<p>User ID <input NAME="eci_userid" VALUE="" SIZE=8>
Password <input Type="password" NAME="eci_password" VALUE="" SIZE=8> <P>
<input type="hidden" NAME="cicsweb_function" VALUE=login><P>
<br>
Please enter your user ID and password,
then press the function you wish to perform<p>
<input type="submit" name="submit" value="Store Object">
<input type="submit" name="submit" value="Delete Object">
<input type="submit" name="submit" value="List Objects">
<input type="reset">
</form>
</body></html>
```

B.2 CICSWEB.HTM: Login HTML Document for Use with GoServe

```
<!doctype html public "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>CICSWEB Administration Login</title></head>
<body>
<img src="cicsweb/cwmast.gif" align=center><p>
<h1>CICSWEB Administration Login</h1>
<form action="$cicsweb" method="POST">
<p>User ID <input NAME="eci_userid" VALUE="" SIZE=8>
Password <input Type="password" NAME="eci_password" VALUE="" SIZE=8> <P>
<input type="hidden" NAME="cicsweb_function" VALUE=login><P>
<br>
Please enter your user ID and password,
then press the function you wish to perform<p>
<input type="submit" name="submit" value="Store Object">
<input type="submit" name="submit" value="Delete Object">
<input type="submit" name="submit" value="List Objects">
<input type="reset">
</form>
</body></html>
```

B.3 CICSWEB.CMD: REXX CGI Script for Use with the IBM Internet Connection Server

```
/*********************************************************************/
/* CGI-BIN script to invoke sample CICS-WEB application examples     */
/* Invokes RxECI to call specified application                       */
/*                                                                   */
/*********************************************************************/
verb = value('REQUEST_METHOD',,'OS2ENVIRONMENT')
/* '@CGIUTILS -ct text/html' */
objid = value('PATH_INFO',,'OS2ENVIRONMENT')     /* Object to display */
objid = substr(objid,2)                          /* strip leading '/' */
if verb = 'POST' then do              /* Only support POSTs for forms */
  /* get the incoming data */
  eciparms = charin(,,value('CONTENT_LENGTH',,'OS2ENVIRONMENT'))
  /* Parse input vars from form into var.x  */
  call parsevar eciparms
end
else   /* not a form, so assume a request to retrieve and display obj */
  return ProcessRetrieveObj(objid)

/* Check if new login attempt */
if translate(var.cicsweb_function) = 'LOGIN' then do   /* new session */
  state = startsession()
  if state \= '' then           /* display any error message returned */
    return response('badreq', state)
end /* LOGIN */
else do                 /* Check that we have a valid current session */
  session = checksession()
  if session \= '' then         /* display any error message returned */
    return response('badreq', session)
end /* check session status */

/* Check if processing an incoming form, or a request for a new form  */
/* - each form has a hidden field (cicsweb_function) with a value that*/
/*   can be checked against the SUBMIT function the user has selected */
/*   If the user presses ENTER, default to process as if SUBMIT button*/
/*   for the form was clicked.                                       */
if var.submit = '' then                     /* User has pressed ENTER */
  var.submit = var.cicsweb_function
if var.submit \= var.cicsweb_function then do     /* request new form */
  /* Now build the document for requested function and send to user   */
  select
    when var.submit = "Store Object" then
      return BuildStoreObj('')
    when var.submit = "List Objects" then
      return BuildListObj('', '')
    when var.submit = "Delete Object" then
      return BuildDeleteObj('')
    when var.submit = "QUIT" then do
      return QuitWeb(var.handle)
      end
    otherwise return response('badreq', 'asked for unknown form')
  end /* select form to display */
end /* build new form */
else do                                  /* Process input from form */
  /* Now build the document for requested function and send to user   */
  select
    when var.submit = "Store Object" then
      return ProcessStoreObj()
    when var.submit = "List Objects" then
      return ProcessListObj()
    when var.submit = "Delete Object" then
      return ProcessDeleteObj()
    otherwise return response('badreq', 'asked for unknown form')
  end /* Process Form Input */
end /* build new form */

return say 'We should never get here'

/*********************************************************************/
/* BuildDeleteObj: Form to delete object stored in CICS              */
/*********************************************************************/
BuildDeleteObj: procedure expose var. eci.
  parse arg message
  title    = 'Delete HTTP Object stored in CICS'
  function = 'Delete Object'
```

```
  if message = '' then                  /* generate default message */
    message  = 'Please enter information and click button for function to perform'
  call Build_header title, function, var.handle, var.eci_userid, message
  say 'CICS Object Name<p>'
  say '<input NAME="CW_Objid" SIZE=48 value="'||var.cw_Objid||'"><p>'
  call Build_footer
  return ''

/*********************************************************************/
/* BuildListObj: Form to list objects stored in CICS                 */
/*********************************************************************/
BuildListObj: procedure expose var. eci.
  parse arg message, list
  title    = 'List HTTP Objects stored in CICS'
  function = 'List Objects'
  if message = '' then                  /* generate default message */
    message  = 'Please enter information and click button for function to perform'
  call Build_header title, function, var.handle, var.eci_userid, message
  say 'CICS Object Name (leave blank for all objects)<p>'
  say '<input NAME="CW_Objkey" SIZE=48 value="'||var.cw_Objkey||'"><p>'
  if list \= '' then
    say list
  call Build_footer
  return ''

/*********************************************************************/
/* BuildStoreObj: Form to store object in CICS                       */
/*********************************************************************/
BuildStoreObj: procedure expose var. eci.
  parse arg message
  title    = 'Store HTTP Object into CICS'
  function = 'Store Object'
  if var.cw_file_name = '' then var.cw_file_name = extract(datadir)
  if message = '' then                  /* generate default message */
    message  = 'Please enter information and click button for function to perform'
  call Build_header title, function, var.handle, var.eci_userid, message
  say 'Workstation File Name (as known on Web Server)<p>'
  say '<input NAME="cw_file_name" SIZE=48 value="'||var.cw_file_name||'"><p>'
  say 'Object Type'
  say '<select NAME="cw_mime_type"><p>'
  say '  <option>application/octet-stream'
  say '  <option>application/postscript'
  say '  <option>application/zip'
  say '  <option>audio/basic'
  say '  <option>audio/x-wav'
  say '  <option>audio/x-midi'
  say '  <option>image/gif'
  say '  <option>image/bmp'
  say '  <option>image/jpeg'
  say '  <option>image/tiff'
  say '  <option selected>text/html'
  say '  <option>text/plain'
  say '  <option>video/mpeg'
  say '  <option>video/x-msvideo'
  say '</select><p>'
  say 'CICS Object Name<p>'
  say '<input NAME="CW_Objid" SIZE=48 value="'||var.cw_Objid||'"><p>'
  say 'Object Description '
  say '<input NAME="CW_Objdesc" SIZE=30 value="'||var.cw_Objdesc||'"><p>'
  call Build_footer
  return ''

/*********************************************************************/
/* Build_Header: Build document and form header for html document    */
/*********************************************************************/
Build_Header: procedure
  parse arg title, function, handle, userid, message
  /* Call CGIUTILS to build HTML Header */
  '@cgiutils -ct text/html'
  say '<!doctype html public "-//IETF//DTD HTML 2.0//EN">'
  say '<html><head>'
  say '<title>'title'</title></head>'
  say '<body>'
  say '<img src="../icons/cwmast.gif" align=center><p>'
  say '<h1>'title'</h1>'
  say '<hr>'message'<hr>'
  say '<form action="/cgi-bin/cicsweb" method="POST">'
  say '<input type="hidden" NAME="cicsweb_function" VALUE="'||function||'">'
```

```
  say '<input type="hidden" NAME="handle" VALUE="'||handle||'">'
  say '<input type="hidden" NAME="eci_userid" VALUE="'||userid||'">'
  return ''

/*********************************************************************/
/* Build_Footer: Build document and form footer for html document    */
/*********************************************************************/
Build_Footer: procedure
  say '<input type="submit" name="submit" value="Store Object">'
  say '<input type="submit" name="submit" value="Delete Object">'
  say '<input type="submit" name="submit" value="List Objects">'
  say '<input type="reset">'
  say '<input type="submit" name="submit" value="QUIT">'
  say '</form></body></html>'
  return ''

/*********************************************************************/
/* ECIMSGTEXT: Display Error Message from ECI Call                   */
/*           e.g. variable eci_userid placed in var.eci_userid       */
/*********************************************************************/
ecimsgtext: procedure  expose ECI.
  msgfile = "c:\ca31\rxcics\rxeci.msg"
  msgtext = 'Unknown ECI return code'
  do until lines(msgfile) = 0
    line = linein(msgfile)
    parse var line returncode message
    if ECI.return_code = returncode then msgtext = message
  end
  rc = lineout(msgfile) /* close file */
  return msgtext

/*********************************************************************/
/* PARSEVAR: Extract variables from form                             */
/*           e.g. variable eci_userid placed in var.eci_userid       */
/*********************************************************************/
parsevar: procedure expose var.
  /* set VAR.x variables from the incoming data stream */
  parse arg data
  VAR.='' /* null string unless set */
  data=data'&' /* set end condition */
  do until data='' /* each value */
    parse var data assign '&' data /* split off name=value */
    parse var assign name '=' value /* separate */
    value=translate(value, ' ', '2b090a0d'x) /* handle '+', tabs, and CRLF */
    name=translate(packur(name)) /* caseless name */
    var.name=packur(value) /* set, after URI decoding */
    /* say name 'is' value */
  end
  return ''

/*********************************************************************/
/* ProcessDeleteObj: Process Form to delete object stored in CICS    */
/*********************************************************************/
ProcessDeleteObj: procedure expose var. eci.
  if var.cw_objid = '' then
    return BuildDeleteObj('Please specify the name of the object you wish to delete')

  /* register RxECI functions */
  rc = rxfuncdrop('ECI')
  If rxfuncquery('ECI') Then Do
     If rxfuncadd('ECI', 'RXECI', 'RxECI' ) Then
       return BuildDeleteObj('Error initializing RxECI')
  End

  /* Initialize ECI call parameters */
  eci.userid   = var.eci_userid
  eci.password = var.eci_password
  eci.program_name = 'CWSERVER'
  eci.extend_mode = "ECI_NO_EXTEND"    /* Only one call to ECI in LUW */

  /* Perform ECI call to delete object */
  var.eci_commarea = left(var.submit,16)||,              /* Function */
                     'CWFILE  '||,                  /* CICS File Name */
                     left(var.cw_objid,64)||,    /* Object ID in CICS */
                     '001'||,                       /* Segment Seq No */
                     copies(' ',80)||,             /* Return msg text */
                     copies('00'x,82)      /* Pad out COMMAREA header */
```

```
  /* Call the program via ECI */
  var.eci_commarea = ECI('ECI.', var.eci_commarea)
  if eci.return_code \= 0 then
    return BuildDeleteObj('ECI Error. RC ='eci.return_code EciMsgText() ECI.abend_code)
  else                      /* Display message returned from CWSERVER */
    return BuildDeleteObj(substr(var.eci_commarea,92,80))

/*********************************************************************/
/* ProcessListObj: Process Form to list objects stored in CICS       */
/*********************************************************************/
ProcessListObj: procedure expose var. eci.
  /* register RxECI functions */
  If rxfuncquery('ECI') Then Do
     If rxfuncadd('ECI', 'RXECI', 'RxECI' ) Then
       return BuildListObj('Error initializing RxECI')
  End

  /* Initialize ECI call parameters */
  eci.userid   = var.eci_userid
  eci.password = var.eci_password
  eci.program_name = 'CWLIST'
  eci.extend_mode = "ECI_NO_EXTEND"    /* Only one call to ECI in LUW */

  /* Perform ECI call to list objects */
  var.eci_commarea = left(var.submit,16)||,              /* Function */
                     'CWFILE  '||,                  /* CICS File Name */
                     left(var.cw_objkey,64)||,   /* Object ID in CICS */
                     '001'||,                       /* Segment Seq No */
                     copies(' ',80)||,             /* Return msg text */
                     copies(' ',30)||,               /* Obj MIME type */
                     right(length(var.cw_objkey),2,'0')||, /* Length of generic key */
                     '00008'||,           /* Length of data following */
                     'cicsweb/'||,  /* Prepend string for anchor href */
                     copies('00'x,31992)          /* Pad out COMMAREA */

  /* Call the program via ECI */
  var.eci_commarea = ECI('ECI.', var.eci_commarea)
  if eci.return_code \= 0 then
    return BuildListObj('ECI Error. RC ='eci.return_code EciMsgText() ECI.abend_code, '')
  else do                   /* Display message returned from CWSERVER */
    listtxt = substr(var.eci_commarea,209,substr(var.eci_commarea,204,5))
    return BuildListObj(substr(var.eci_commarea,92,80),listtxt)
  end

/*********************************************************************/
/* ProcessRetrieveObj: Process Browser Retrieve Request              */
/*********************************************************************/
ProcessRetrieveObj: procedure expose var. eci.
  parse arg objid
  /* register RxECI functions */
  If rxfuncquery('ECI') Then Do
     If rxfuncadd('ECI', 'RXECI', 'RxECI' ) Then
       return response('badreq','Error initializing RxECI')
  End

  /* Initialize ECI call parameters */
  /* Need to specify userid and password since not passed from form */
  eci.userid   = 'DEFAULT'
  eci.password = 'DEFAULT'
  eci.program_name = 'CWSERVER'
  eci.extend_mode = "ECI_NO_EXTEND" /* Read-only, so each call is LUW */

  /* Perform ECI call to get first or only segment of object */
  segsize = 8000    /* Object will be passed in segments of this size */
  var.eci_commarea = left('Retrieve Object',16)||,        /* Function */
                     'CWFILE  '||,                  /* CICS File Name */
                     left(objid,64)||,           /* Object ID in CICS */
                     '001'||,                       /* Segment Seq No */
                     copies(' ',80)||,             /* Return msg text */
                     copies(' ',30)||,               /* Obj MIME type */
                     '000'||,                      /* No. of Segments */
                     '0000000'||,                   /* Total Obj Size */
                     '00000'||,                  /* this segment size */
                     copies(' ',30)||,          /* Object Description */
                     copies('00'x,segsize)             /* Object Data */

  /* Call the program via ECI */
  var.eci_commarea = ECI('ECI.', var.eci_commarea)
```

```
  if eci.return_code \= 0 then do
    msg = 'resulted in ECI Error. RC ='eci.return_code EciMsgText() ECI.abend_code
    return response('badreq', msg)
  end
  if substr(var.eci_commarea,92,80) \= ' 'then do /* Error msg returned */
    return response('badreq','resulted in 'substr(var.eci_commarea,92,80))
  end

  /* Extract what we need from commarea */
  mimetype  = substr(var.eci_commarea,172,30)
  segments  = substr(var.eci_commarea,202,3)
  seglength = substr(var.eci_commarea,212,5)    /* length of this seg */
  objdata   = substr(var.eci_commarea,247,seglength)
  objsize   = substr(var.eci_commarea,205,7)
  /* Call CGIUTILS to build HTML Header */
  '@cgiutils -ct 'mimetype '-length ' objsize
  /* If only one segment, return it immediately */
  if segments = 1 then do
    rc = charout(,objdata)                          /* Send the data */
    return ''
  end
  else do /* return first seg, then go retrieve rest */
    rc = charout(,objdata)                     /* Send first segment */
    do i = 2 to segments           /* loop to get remaining segments */
      var.eci_commarea = left('Retrieve Object',16)||,    /* Function */
                         'CWFILE  '||,             /* CICS File Name */
                         left(objid,64)||,       /* Object ID in CICS */
                         right(i,3,'0')||,          /* Segment Seq No */
                         copies(' ',80)||,         /* Return msg text */
                         copies(' ',30)||,           /* Obj MIME type */
                         '000'||,                  /* No. of Segments */
                         '0000000'||,               /* Total Obj Size */
                         '00000'||,              /* this segment size */
                         copies(' ',30)||,      /* Object Description */
                         copies('00'x,segsize)         /* Object Data */

      /* Call the program via ECI */
      var.eci_commarea = ECI('ECI.', var.eci_commarea)
      if eci.return_code \= 0 then do /* attempt to send msg and quit */
        say '<p>Request resulted in ECI Error. RC ='
        say msg||eci.return_code EciMsgText() ECI.abend_code'<p>'
        return ''                                       /* bail out */
      end
      if substr(var.eci_commarea,92,80) \= ' 'then do /* Error msg returned */
        say '<p>Request returned error message' substr(var.eci_commarea,92,80)'<p>'
        return ''                                       /* bail out */
      end
      /* Extract what we need from commarea */
      mimetype  = substr(var.eci_commarea,172,30)
      seglength = substr(var.eci_commarea,212,5)
      objdata   = substr(var.eci_commarea,247,seglength)
      rc = charout(,objdata)                    /* Send this segment */
    end
    return ''
  end

/*********************************************************************/
/* ProcessStoreObj: Process Form to store object in CICS             */
/*********************************************************************/
ProcessStoreObj: procedure expose var. eci.
  /* check if specified file exists and contains data */
  if stream(var.cw_file_name,'c','query exists') = '' then
    return BuildStoreObj('File <em> 'var.cw_file_name' </em>does not exist')
  objlength = stream(var.cw_file_name, 'c', 'query size')
  if objlength = 0 thendo
    msg = '<em> 'var.cw_file_name' </em>contains no data.  Request not processed'
    return BuildStoreObj(msg)
  end
  if var.cw_objid = '' then do
    parse var var.cw_file_name . ':' var.cw_objid /* strip drive letter */
    if var.cw_objid = '' then var.cw_objid = var.cw_file_name
    msg = 'Verify CICS object name and click <strong> Store Object</strong>'
    return BuildStoreObj(msg)
  end
  if var.cw_objdesc = '' then do
    return BuildStoreObj('Please provide a Description of the object')
  end
```

```
  /* Get the object contents */
  objdata = charin(var.cw_file_name,1,objlength)

  /* register RxECI functions */
  /* rc = rxfuncdrop('ECI') */
  If rxfuncquery('ECI') Then Do
     If rxfuncadd('ECI', 'RXECI', 'RxECI' ) Then
       return BuildStoreObj('Error initializing RxECI')
  End

  /* Initialize ECI call parameters */
  eci.userid   = var.eci_userid
  eci.password = var.eci_password
  eci.program_name = 'CWSERVER'
  eci.extend_mode = "ECI_EXTENDED"          /* Multiple calls in LUW */

  /* Determine how many times we need to call ECI to pass the data */
  segsize = 8000    /* Object will be passed in segments of this size */
  segments = ((objlength-1) % segsize) + 1
  /* loop to send segments to CICS */
  do i = 1 to segments
    /* Build the COMMAREA */
    if i = segments then do
      seglength = (objlength // segsize)
/*    eci.extend_mode = "ECI_NO_EXTEND"*/ /* This is last call in LUW */
    end
    else seglength = segsize       /* length of segment for this call */
    var.eci_commarea = left(var.submit,16)||,             /* Function */
                       'CWFILE  '||,               /* CICS File Name */
                       left(var.cw_objid,64)||,  /* Object ID in CICS */
                       right(i,3,'0')||,            /* Segment Seq No */
                       copies(' ',80)||,           /* Return msg text */
                       left(var.cw_mime_type,30)||,  /* Obj MIME type */
                       right(segments,3,'0')||,    /* No. of Segments */
                       right(objlength,7,'0')||,    /* Total Obj Size */
                       right(seglength,5,'0')||, /* this segment size */
                       left(var.cw_objdesc,30)||,  /* Obj Description */
                       substr(objdata,(i-1)*segsize+1,segsize,'00'x)
                                                    /* Object Data */

    /* Call the program via ECI */
    var.eci_commarea = ECI('ECI.', var.eci_commarea)
    if eci.return_code \= 0 then
      return BuildStoreObj('ECI Error. RC ='eci.return_code EciMsgText() ECI.abend_code)
  end
  /* Commit LUW.  Separate call reqd when using ECI direct on CICS    */
  /* OS/2 Server.  Normally simply changing extend_mode to            */
  /* ECI_NO_EXTEND on last call should be sufficient to commit LUW    */
  eci.program_name = ''
  eci.extend_mode = "ECI_COMMIT"                          /* End LUW */
  var.eci_commarea = ECI('ECI.', var.eci_commarea)
  if eci.return_code \= 0 then
    return BuildStoreObj('ECI Error. RC ='eci.return_code EciMsgText() ECI.abend_code)
  return BuildStoreObj(substr(var.eci_commarea,92,80))

/*********************************************************************/
/* QuitWeb: Delete State File and Return to Login Screen             */
/*********************************************************************/
QuitWeb: procedure
  parse arg handle
  port = extract('serverport')
  '@del tmp\CW'||handle||'.'||port
  '@cgiutils -ct text/html -status 302 -uri cicsweb.html -noel'
  if port = 80 then port = ''
  else port = ':'||port
  say 'Location: http://'extract(serveraddr)||port'/cicsweb.html'
  say ''
  return ''

/*********************************************************************/
/* STARTSESSION: build server session-state file                     */
/*********************************************************************/
startsession: procedure expose var. eci.
  if (var.eci_userid = '') | (var.eci_password = '') then
    return 'needs to include user ID and password'
  var.handle = extract(transaction)          /* Save for later use   */
  /* generate new state file name */
  statefile = 'tmp/CW'||var.handle'.'extract(serverport)
```

```
  call lineout statefile, var.eci_userid, 1   /* userid to state file */
  call lineout statefile, var.eci_password    /* pw to state file     */
  return ''

/*********************************************************************/
/* CHECKSESSION: validate request against existing state file        */
/*   Checks that userid in state file matches userid sent from form  */
/*********************************************************************/
checksession: procedure expose var. eci.
  /* generate state file name */
  statefile = 'tmp/CW'||var.handle'.'extract(serverport)
  if stream(statefile, 'c', 'query exists') = '' then    /* Not found */
    return 'No current session exists.  Please login'
  if linein(statefile, 1) \= var.eci_userid then
    return "We don't have a record of your user id.  Please login again"
  var.eci_password = linein(statefile)      /* get pw from state file */
  rc = stream(statefile, 'c', 'close')      /* Close the state file   */
  return ''

/*********************************************************************/
/* The following functions are designed to be compatible with GoServe */
/* functions, and have been written to facilitate porting GoServe    */
/* filters to use the cgi-bin interface.                             */
/*********************************************************************/

/*********************************************************************/
/* Clientname: Return server name (modelled after GoServe)           */
/*********************************************************************/
Clientname: procedure
  return value('REMOTE_HOST',,'OS2ENVIRONMENT')

/*********************************************************************/
/* Extract: Return value of various settings                         */
/*********************************************************************/
extract: procedure
  parse upper arg var
  select
    when var = "DATADIR" then
      return 'C:\WWW\HTML\'
    when var = "CLIENTADDR" then
      return value('REMOTE_ADDR',,'OS2ENVIRONMENT')
    when var = "SERVERADDR" then
      return value('SERVER_NAME',,'OS2ENVIRONMENT')
    when var = "SERVERPORT" then
      return value('SERVER_PORT',,'OS2ENVIRONMENT')
    when var = "SERVERPROTOCOL" then
      return value('SERVER_PROTOCOL',,'OS2ENVIRONMENT')
    when var = "SERVERSOFTWARE" then
      return value('SERVER_SOFTWARE',,'OS2ENVIRONMENT')
    when var = "TRANSACTION" then do          /* get a session handle */
      /* return a quasi-unique number for building a session handle   */
      statefile = 'cwhandle.seq'
      handle = 1
      if stream(statefile, 'c', 'query exists') = '' then /* Not found*/
        call lineout statefile, '1', 1             /* Initialise file */
      else do                    /* Increment handle sequence number */
        handle = linein(statefile, 1) + 1
        call lineout statefile, handle, 1              /* Update file */
      end
      rc = stream(statefile, 'c', 'close')     /* Close the seq. file */
      return handle
    end
    otherwise return 'UNKNOWN'
  end

/*********************************************************************/
/* Packur:                                                           */
/* This procedure takes an input string, and converts characters     */
/* encoded as escape sequences (e.g. %xx) back to the character that */
/* they represent.                                                   */
/*********************************************************************/
packur: procedure
  parse arg outstring '%' rest
  do while length(rest) > 1
    outstring = outstring||x2c(substr(rest,1,2))      /* decode value */
    rest = substr(rest,3)
    parse var rest next '%' rest
    outstring = outstring||next
```

```
  end
  return outstring

/*********************************************************************/
/* Response: Send an error response to client                        */
/*********************************************************************/
response: procedure
  parse arg request, message
  select
    when request='badreq'   then use='400 Bad Request Syntax'
    when request='notfound' then use='404 Not found'
    when request='forbid'   then use='403 Forbidden'
    when request='unauth'   then use='401 Unauthorized'
  end  /* Add others to this list as needed */
  /* Now set the response and build the response file */
  parse var use code text
  '@cgiutils -ct text/html -status ' code
  say '<!doctype html public "-//IETF//DTD HTML 2.0//EN">'
  say "<html><head><title>"text"</title></head>"
  say "<body><h2>Sorry...</h2>"
  say "<p>The request from your Web client" message"."
  say "<hr><em>HTTP response code:</em>" code '['text']'
  say "<br><em>From server at:</em>" servername()
  say "<br><em>Running:</em>" server()
  say "</body></html>"
  return ''                                 /* [called as function] */


/*********************************************************************/
/* Servername: Return server name (modelled after GoServe)           */
/*********************************************************************/
Servername: procedure
  return value('SERVER_NAME',,'OS2ENVIRONMENT')

/*********************************************************************/
/* Server: Return server software (modelled after GoServe)           */
/*********************************************************************/
Server: procedure
  return value('SERVER_SOFTWARE',,'OS2ENVIRONMENT')
```

B.4 CICSWEB.80: REXX Filter for Use with GoServe

```
/*********************************************************************/
/* GoServe filter to invoke sample CICS-WEB application examples     */
/* Invokes RxECI to call specified application                       */
/*                                                                   */
/* Arguments:                                                        */
/*   verb  - how called (only POST supported)                        */
/*   objid - selector after cgi-bin/cicsweb/ stripped from front     */
/*           (for retrieval, identifies object to retrieve)          */
/*********************************************************************/
parse arg verb, objid
if verb = 'POST' then do              /* Only support POSTs for forms */
  'read body var formparms'                   /* get the incoming data */
  if rc=-4 then                                    /* body too large */
    return response('badreq', 'sent too much data')
  if rc<>0 then                          /* e.g., invalid HTTP header */
    return response('badreq', 'sent data that could not be read')
  /* Parse input vars from form into var.x */
  call parsevar(formparms)
end
else   /* not a form, so assume a request to retrieve and display obj */
  return ProcessRetrieveObj(objid)

/* Check if new login attempt */
if translate(var.cicsweb_function) = 'LOGIN' then do   /* new session */
  state = startsession()
  if state \= '' then           /* display any error message returned */
    return response('badreq', state)
end /* LOGIN */
else do                 /* Check that we have a valid current session */
  session = checksession()
  if session \= '' then         /* display any error message returned */
    return response('badreq', session)
end /* check session status */

/* Check if processing an incoming form, or a request for a new form  */
/* - each form has a hidden field (cicsweb_function) with a value that*/
/*   can be checked against the SUBMIT function the user has selected */
/*   If the user presses ENTER, default to process as if SUBMIT button*/
/*   for the form was clicked.                                       */
if var.submit = '' then                    /* User has pressed ENTER */
  var.submit = var.cicsweb_function
if var.submit \= var.cicsweb_function then do     /* request new form */
  /* Now build the document for requested function and send to user   */
  select
    when var.submit = "Store Object" then
      return BuildStoreObj('')
    when var.submit = "List Objects" then
      return BuildListObj('', '')
    when var.submit = "Delete Object" then
      return BuildDeleteObj('')
    when var.submit = "QUIT" then do
      return QuitWeb(var.handle)
      end
    otherwise return response('badreq', 'asked for unknown form')
  end /* select form to display */
end /* build new form */
else do                                  /* Process input from form */
  /* Now build the document for requested function and send to user   */
  select
    when var.submit = "Store Object" then
      return ProcessStoreObj()
    when var.submit = "List Objects" then
      return ProcessListObj()
    when var.submit = "Delete Object" then
      return ProcessDeleteObj()
    otherwise return response('badreq', 'asked for unknown form')
  end /* Process Form Input */
end /* build new form */

return say 'We should never get here'

/*********************************************************************/
/* BuildDeleteObj: Form to delete object stored in CICS              */
/*********************************************************************/
BuildDeleteObj: procedure expose var. eci.
```

```
parse arg message
title    = 'Delete HTTP Object stored in CICS'
function = 'Delete Object'
if message = '' then                     /* generate default message */
  message  = 'Please enter information and click button for function to perform'
crlf='0d0a'x
html = ''
html = Build_header(html, title, function, var.handle, var.eci_userid, message)
html = html||'CICS Object Name<p>'crlf
html = html||'<input NAME="CW_Objid" SIZE=48 value="'||var.cw_Objid||'"><p>'crlf
html = Build_footer(html)
'VAR TYPE text/html AS DELOBJ NAME html'
return ''

/*********************************************************************/
/* BuildListObj: Form to list objects stored in CICS                 */
/*********************************************************************/
BuildListObj: procedure expose var. eci.
parse arg message, list
title    = 'List HTTP Objects stored in CICS'
function = 'List Objects'
if message = '' then                     /* generate default message */
  message  = 'Please enter information and click button for function to perform'
crlf='0d0a'x
html = ''
html = Build_header(html, title, function, var.handle, var.eci_userid, message)
html = html||'CICS Object Name (leave blank for all objects)<p>'crlf
html = html||'<input NAME="CW_Objkey" SIZE=48 value="'||var.cw_Objkey||'"><p>'crlf
if list \= '' then
  html = html||list
html = Build_footer(html)
'VAR TYPE text/html AS LISTOBJ NAME html'
return ''

/*********************************************************************/
/* BuildStoreObj: Form to store object in CICS                       */
/*********************************************************************/
BuildStoreObj: procedure expose var. eci.
parse arg message
title    = 'Store HTTP Object into CICS'
function = 'Store Object'
if var.cw_file_name = '' then var.cw_file_name = extract(datadir)
if message = '' then                     /* generate default message */
  message  = 'Please enter information and click button for function to perform'
crlf='0d0a'x
html = ''
html = Build_header(html, title, function, var.handle, var.eci_userid, message)
html = html||'Workstation File Name (as known on Web Server)<p>'crlf
html = html||'<input NAME="cw_file_name" SIZE=48 value="'||var.cw_file_name||'"><p>'crlf
html = html||'Object Type'crlf
html = html||'<select NAME="cw_mime_type"><p>'crlf
html = html||'  <option>application/octet-stream'crlf
html = html||'  <option>application/postscript'crlf
html = html||'  <option>application/zip'crlf
html = html||'  <option>audio/basic'crlf
html = html||'  <option>audio/x-wav'crlf
html = html||'  <option>audio/x-midi'crlf
html = html||'  <option>image/gif'crlf
html = html||'  <option>image/bmp'crlf
html = html||'  <option>image/jpeg'crlf
html = html||'  <option>image/tiff'crlf
html = html||'  <option selected>text/html'crlf
html = html||'  <option>text/plain'crlf
html = html||'  <option>video/mpeg'crlf
html = html||'  <option>video/x-msvideo'crlf
html = html||'</select><p>'crlf
html = html||'CICS Object Name<p>'crlf
html = html||'<input NAME="CW_Objid" SIZE=48 value="'||var.cw_Objid||'"><p>'crlf
html = html||'Object Description 'crlf
html = html||'<input NAME="CW_Objdesc" SIZE=30 value="'||var.cw_Objdesc||'"><p>'crlf
html = Build_footer(html)
'VAR TYPE text/html AS STOROBJ NAME html'
return ''

/*********************************************************************/
/* Build_Header: Build document and form header for html document    */
/*********************************************************************/
Build_Header: procedure
```

```
parse arg html, title, function, handle, userid, message
crlf='0d0a'x
html = html||'<!doctype html public "-//IETF//DTD HTML 2.0//EN">'crlf
html = html||'<html><head>'crlf
html = html||'<title>'title'</title></head>'crlf
html = html||'<body>'crlf
html = html||'<img src="cicsweb/cwmast.gif" align=center><p>'crlf
html = html||'<h1>'title'</h1>'crlf
html = html||'<hr>'message'<hr>'crlf
html = html||'<form action="$cicsweb" method="POST">'crlf
html = html||'<input type="hidden" NAME="cicsweb_function" VALUE="'||function||'">'crlf
html = html||'<input type="hidden" NAME="handle" VALUE="'||handle||'">'crlf
html = html||'<input type="hidden" NAME="eci_userid" VALUE="'||userid||'">'crlf
return html

/*********************************************************************/
/* Build_Footer: Build document and form footer for html document    */
/*********************************************************************/
Build_Footer: procedure
parse arg html
crlf='0d0a'x
html = html||'<input type="submit" name="submit" value="Store Object">'crlf
html = html||'<input type="submit" name="submit" value="Delete Object">'crlf
html = html||'<input type="submit" name="submit" value="List Objects">'crlf
html = html||'<input type="reset">'crlf
html = html||'<input type="submit" name="submit" value="QUIT">'crlf
html = html||'</form></body></html>'crlf
return html

/*********************************************************************/
/* ECIMSGTEXT: Display Error Message from ECI Call                   */
/*           e.g. variable eci_userid placed in var.eci_userid       */
/*********************************************************************/
ecimsgtext: procedure  expose ECI.
  msgfile = "c:\ca31\rxcics\rxeci.msg"
  msgtext = 'Unknown ECI return code'
  do until lines(msgfile) = 0
    line = linein(msgfile)
    parse var line returncode message
    if ECI.return_code = returncode then msgtext = message
  end
  rc = lineout(msgfile) /* close file */
return msgtext

/*********************************************************************/
/* PARSEVAR: Extract variables from form                             */
/*           e.g. variable eci_userid placed in var.eci_userid       */
/*********************************************************************/
parsevar: procedure expose var.
  /* set VAR.x variables from the incoming data stream */
  parse arg data
  VAR.=''                                  /* null string unless set */
  data=data'&'                             /* set end condition */
  do until data=''                         /* each value */
    parse var data assign '&' data         /* split off name=value */
    parse var assign name '=' value        /* separate */
    value=translate(value, ' ', '2b090a0d'x) /* handle '+', tabs, and CRLF */
    name=translate(packur(name))           /* caseless name */
    var.name=packur(value)                 /* set, after URI decoding */
    /* say name 'is' value */
  end
  return ''

/*********************************************************************/
/* ProcessDeleteObj: Process Form to delete object stored in CICS    */
/*********************************************************************/
ProcessDeleteObj: procedure expose var. eci.
  if var.cw_objid = '' then
    return BuildDeleteObj('Please specify the name of the object you wish to delete')

  /* register RxECI functions */
  rc = rxfuncdrop('ECI')
  If rxfuncquery('ECI') Then Do
     If rxfuncadd('ECI', 'RXECI', 'RxECI' ) Then
       return BuildDeleteObj('Error initializing RxECI')
  End

  /* Initialize ECI call parameters */
```

```
  eci.userid   = var.eci_userid
  eci.password = var.eci_password
  eci.program_name = 'CWSERVER'
  eci.extend_mode = "ECI_NO_EXTEND"    /* Only one call to ECI in LUW */

  /* Perform ECI call to delete object */
  var.eci_commarea = left(var.submit,16)||,              /* Function */
                     'CWFILE  '||,                 /* CICS File Name */
                     left(var.cw_objid,64)||,    /* Object ID in CICS */
                     '001'||,                      /* Segment Seq No */
                     copies(' ',80)||,            /* Return msg text */
                     copies('00'x,82)      /* Pad out COMMAREA header */

  /* Call the program via ECI */
  var.eci_commarea = ECI('ECI.', var.eci_commarea)
  if eci.return_code \= 0 then
    return BuildDeleteObj('ECI Error. RC ='eci.return_code EciMsgText() ECI.abend_code)
  else                      /* Display message returned from CWSERVER */
    return BuildDeleteObj(substr(var.eci_commarea,92,80))

/*********************************************************************/
/* ProcessListObj: Process Form to list objects stored in CICS       */
/*********************************************************************/
ProcessListObj: procedure expose var. eci.
  /* register RxECI functions */
  If rxfuncquery('ECI') Then Do
     If rxfuncadd('ECI', 'RXECI', 'RxECI' ) Then
       return BuildListObj('Error initializing RxECI')
  End

  /* Initialize ECI call parameters */
  eci.userid   = var.eci_userid
  eci.password = var.eci_password
  eci.program_name = 'CWLIST'
  eci.extend_mode = "ECI_NO_EXTEND"    /* Only one call to ECI in LUW */

  /* Perform ECI call to list objects */
  var.eci_commarea = left(var.submit,16)||,              /* Function */
                     'CWFILE  '||,                 /* CICS File Name */
                     left(var.cw_objkey,64)||,   /* Object ID in CICS */
                     '001'||,                      /* Segment Seq No */
                     copies(' ',80)||,            /* Return msg text */
                     copies(' ',30)||,              /* Obj MIME type */
                     right(length(var.cw_objkey),2,'0')||, /* Length of generic key */
                     '00009'||,           /* Length of data following */
                     '$cicsweb/'||,/* string to prepend to <a href="..*/
                     copies('00'x,31991)          /* Pad out COMMAREA */

  /* Call the program via ECI */
  var.eci_commarea = ECI('ECI.', var.eci_commarea)
  if eci.return_code \= 0 then
    return BuildListObj('ECI Error. RC ='eci.return_code EciMsgText() ECI.abend_code, '')
  else do                  /* Display message returned from CWSERVER */
    listtxt = substr(var.eci_commarea,209,substr(var.eci_commarea,204,5))
    return BuildListObj(substr(var.eci_commarea,92,80),listtxt)
  end

/*********************************************************************/
/* ProcessRetrieveObj: Process Browser Retrieve Request              */
/*********************************************************************/
ProcessRetrieveObj: procedure expose var. eci.
  parse arg objid
  /* register RxECI functions */
  If rxfuncquery('ECI') Then Do
     If rxfuncadd('ECI', 'RXECI', 'RxECI' ) Then
       return response('badreq','Error initializing RxECI')
  End

  /* Initialize ECI call parameters */
  /* Need to specify userid and password since not passed from form */
  eci.userid   = 'DEFAULT'
  eci.password = 'DEFAULT'
  eci.program_name = 'CWSERVER'
  eci.extend_mode = "ECI_NO_EXTEND" /* Read-only, so each call is LUW */

  /* Perform ECI call to get first or only segment of object */
  segsize = 8000    /* Object will be passed in segments of this size */
  var.eci_commarea = left('Retrieve Object',16)||,        /* Function */
```

```
                    'CWFILE  '||,                 /* CICS File Name */
                    left(objid,64)||,           /* Object ID in CICS */
                    '001'||,                      /* Segment Seq No */
                    copies(' ',80)||,           /* Return msg text */
                    copies(' ',30)||,             /* Obj MIME type */
                    '000'||,                    /* No. of Segments */
                    '0000000'||,                  /* Total Obj Size */
                    '00000'||,                 /* this segment size */
                    copies(' ',30)||,          /* Object Description */
                    copies('00'x,segsize)            /* Object Data */

  /* Call the program via ECI */
  var.eci_commarea = ECI('ECI.', var.eci_commarea)
  if eci.return_code \= 0 then do
    msg = 'resulted in ECI Error. RC ='eci.return_code EciMsgText() ECI.abend_code
    return response('badreq', msg)
  end
  if substr(var.eci_commarea,92,80) \= ' 'then do /* Error msg returned */
    return response('badreq','resulted in 'substr(var.eci_commarea,92,80))
  end

  /* Extract what we need from commarea */
  mimetype  = substr(var.eci_commarea,172,30)
  segments  = substr(var.eci_commarea,202,3)
  seglength = substr(var.eci_commarea,212,5)    /* length of this seg */
  objdata   = substr(var.eci_commarea,247,seglength)
  /* If only one segment, return it immediately */
  if segments = 1 then do
    'VAR TYPE 'mimetype' AS CWRetrieve NAME objdata'
    return ''
  end
  else do /* return first seg, then go retrieve rest */
    'SEND TYPE 'mimetype' AS CWRetrieve'              /* start output */
    'var name objdata'                          /* Send first segment */
    do i = 2 to segments           /* loop to get remaining segments */
      var.eci_commarea = left('Retrieve Object',16)||,    /* Function */
                         'CWFILE  '||,              /* CICS File Name */
                         left(objid,64)||,       /* Object ID in CICS */
                         right(i,3,'0')||,          /* Segment Seq No */
                         copies(' ',80)||,        /* Return msg text */
                         copies(' ',30)||,          /* Obj MIME type */
                         '000'||,                 /* No. of Segments */
                         '0000000'||,               /* Total Obj Size */
                         '00000'||,              /* this segment size */
                         copies(' ',30)||,      /* Object Description */
                         copies('00'x,segsize)         /* Object Data */

      /* Call the program via ECI */
      var.eci_commarea = ECI('ECI.', var.eci_commarea)
      if eci.return_code \= 0 then do /* attempt to send msg and quit */
        msg = 'Request resulted in ECI Error. RC ='
        'STRING' msg||eci.return_code EciMsgText() ECI.abend_code
        'SEND COMPLETE'                       /* Signal all data sent */
        return ''                                    /* bail out */
      end
      if substr(var.eci_commarea,92,80) \= ' 'then do /* Error msg returned */
        'STRING Request returned error message' substr(var.eci_commarea,92,80)
        'SEND COMPLETE'                       /* Signal all data sent */
        return ''                                    /* bail out */
      end
      /* Extract what we need from commarea */
      mimetype  = substr(var.eci_commarea,172,30)
      seglength = substr(var.eci_commarea,212,5)
      objdata   = substr(var.eci_commarea,247,seglength)
      'VAR NAME objdata'                          /* Send this segment */
    end
    'SEND COMPLETE'                           /* Signal all data sent */
    return ''
  end

/*********************************************************************/
/* ProcessStoreObj: Process Form to store object in CICS             */
/*********************************************************************/
ProcessStoreObj: procedure expose var. eci.
  /* check if specified file exists and contains data */
  if stream(var.cw_file_name,'c','query exists') = '' then
    return BuildStoreObj('File <em> 'var.cw_file_name' </em>does not exist')
  objlength = stream(var.cw_file_name, 'c', 'query size')
```

```
  if objlength = 0 then do
    msg = '<em> 'var.cw_file_name' </em>contains no data.  Request not processed'
    return BuildStoreObj(msg)
  end
  if var.cw_objid = '' then do
    parse var var.cw_file_name . ':' var.cw_objid /* strip drive letter */
    if var.cw_objid = '' then var.cw_objid = var.cw_file_name
    return BuildStoreObj('Verify CICS object name and click <strong> Store Object</strong>')
  end
  if var.cw_objdesc = '' then do
    return BuildStoreObj('Please provide a Description of the object')
  end

  /* Get the object contents */
  objdata = charin(var.cw_file_name,1,objlength)

  /* register RxECI functions */
  /* rc = rxfuncdrop('ECI') */
  If rxfuncquery('ECI') Then Do
     If rxfuncadd('ECI', 'RXECI', 'RxECI' ) Then
       return BuildStoreObj('Error initializing RxECI')
  End

  /* Initialize ECI call parameters */
  eci.userid   = var.eci_userid
  eci.password = var.eci_password
  eci.program_name = 'CWSERVER'
  eci.extend_mode = "ECI_EXTENDED"           /* Multiple calls in LUW */

  /* Determine how many times we need to call ECI to pass the data */
  segsize = 8000    /* Object will be passed in segments of this size */
  segments = ((objlength-1) % segsize) + 1
  /* loop to send segments to CICS */
  do i = 1 to segments
    /* Build the COMMAREA */
    if i = segments then do
      seglength = (objlength // segsize)
/*    eci.extend_mode = "ECI_NO_EXTEND"*/ /* This is last call in LUW */
    end
    else seglength = segsize       /* length of segment for this call */
    var.eci_commarea = left(var.submit,16)||,              /* Function */
                       'CWFILE  '||,                /* CICS File Name */
                       left(var.cw_objid,64)||,  /* Object ID in CICS */
                       right(i,3,'0')||,            /* Segment Seq No */
                       copies(' ',80)||,           /* Return msg text */
                       left(var.cw_mime_type,30)||,  /* Obj MIME type */
                       right(segments,3,'0')||,    /* No. of Segments */
                       right(objlength,7,'0')||,    /* Total Obj Size */
                       right(seglength,5,'0')||, /* this segment size */
                       left(var.cw_objdesc,30)||,  /* Obj Description */
                       substr(objdata,(i-1)*segsize+1,segsize,'00'x)
                                                    /* Object Data */

    /* Call the program via ECI */
    var.eci_commarea = ECI('ECI.', var.eci_commarea)
    if eci.return_code \= 0 then
      return BuildStoreObj('ECI Error. RC ='eci.return_code EciMsgText() ECI.abend_code)
  end
  /* Commit LUW.  Separate call reqd when using ECI direct on CICS    */
  /* OS/2 Server.  Normally simply changing extend_mode to            */
  /* ECI_NO_EXTEND on last call should be sufficient to commit LUW    */
  eci.program_name = ''
  eci.extend_mode = "ECI_COMMIT"                          /* End LUW */
  var.eci_commarea = ECI('ECI.', var.eci_commarea)
  if eci.return_code \= 0 then
    return BuildStoreObj('ECI Error. RC ='eci.return_code EciMsgText() ECI.abend_code)
  return BuildStoreObj(substr(var.eci_commarea,92,80))

/*********************************************************************/
/* QuitWeb: Delete State File and Return to Login Screen             */
/*********************************************************************/
QuitWeb: procedure
parse arg handle
address cmd 'del tmp\CW'||handle||'.'||extract(serverport)
return 'FILE NAME '||extract(datadir)||'cicsweb.htm'

/*********************************************************************/
/* RESPONSE: Standard (mostly error) responses (from GoRemote)       */
```

```
/*                                                                   */
/* Arguments are: response type and extended message information.     */
/* It uses the VAR command to send message document to Web Browser    */
/*********************************************************************/
response: procedure
  parse arg request, message
  select
    when request='badreq'   then use='400 Bad Request Syntax'
    when request='notfound' then use='404 Not found'
    when request='forbid'   then use='403 Forbidden'
    when request='unauth'   then use='401 Unauthorized'
  end  /* Add others to this list as needed */
  /* Now set the response and build the response file */
  'RESPONSE HTTP/1.0' use     /* Set HTTP response line */
  parse var use code text
  crlf='0d0a'x; out=''
  out=out||'<!doctype html public "-//IETF//DTD HTML 2.0//EN">'crlf
  out=out||"<html><head><title>"text"</title></head>"crlf
  out=out||"<body><h2>Sorry...</h2>"crlf
  out=out||"<p>The request from your Web client" message"."crlf
  out=out||"<hr><em>HTTP response code:</em>" code '['text']'crlf
  out=out||"<br><em>From server at:</em>" servername()crlf
  out=out||"<br><em>Running:</em>" server()crlf
  out=out||"</body></html>"crlf
  'VAR TYPE text/html AS Response NAME out'  /* send it */
  return ''                                  /* [called as function] */

/*********************************************************************/
/* STARTSESSION: build server session-state file                     */
/*********************************************************************/
startsession: procedure expose var. eci.
  /* Upper case userid and PW */
  var.eci_userid   = translate(var.eci_userid)
  var.eci_password = translate(var.eci_password)
  if (var.eci_userid = '') | (var.eci_password = '') then
    return 'needs to include user ID and password'
  var.handle = extract(transaction)           /* Save for later use   */
  /* generate new state file name */
  statefile = 'tmp/CW'||var.handle'.'extract(serverport)
  call lineout statefile, var.eci_userid, 1   /* userid to state file */
  call lineout statefile, var.eci_password    /* pw to state file     */
  return ''

/*********************************************************************/
/* CHECKSESSION: validate request against existing state file        */
/*   Checks that userid in state file matches userid sent from form  */
/*********************************************************************/
checksession: procedure expose var. eci.
  /* generate state file name */
  statefile = 'tmp/CW'||var.handle'.'extract(serverport)
  if stream(statefile, 'c', 'query exists') = '' then    /* Not found */
    return 'No current session exists.  Please login'
  if linein(statefile, 1) \= var.eci_userid then
    return "We don't have a record of your user id.  Please login again"
  var.eci_password = linein(statefile)      /* get pw from state file */
  rc = stream(statefile, 'c', 'close')      /* Close the state file   */
  return ''
```

B.5 CICS COBOL Program to Store, Retrieve, and Delete Objects

```
000100*****************************************************************
000200*    CWSERVER - CICS Server program for Web Sample              *
000300*****************************************************************
000400*    CREATED 95/08/08 BY David Thrum                            *
000500*                                                               *
000600*    COPYRIGHT - IBM                                            *
000700*                                                               *
000800*    FUNCTION  - Called from Web Server via ECI                 *
000900*              - Performs requested function:                   *
001000*                - Store object passed in COMMAREA in VSAM file *
001100*                - Delete object from VSAM file                 *
001300*                - Retrieve object from VSAM file and return    *
001400*                  in COMMAREA                                  *
001500*    PARAMETERS PASSED                                          *
001600*              - CICS COMMAREA                                  *
001700*                                                               *
001710*    This is an example of a program that is not "HTML-aware".  *
001720*    Any HTML that is required to be sent to the Web Browser    *
001730*    is either contained in the objects managed by this         *
001740*    program, or must be generated at the Web Server by the     *
001750*    CGI script, GoServe filter, or equivalent.                 *
001770*                                                               *
001800*****************************************************************
001900 IDENTIFICATION DIVISION.
002000 PROGRAM-ID. CWSERVER.
002100 ENVIRONMENT DIVISION.
002200 DATA DIVISION.
002300*
002400 WORKING-STORAGE SECTION.
002500*
002600*  MISCELLANEOUS WORK AREAS/CONSTANTS
002700*
002800 77  WS-LOW-VALUES       PIC X VALUE LOW-VALUES.
002830 77  CW-RECLEN           PIC S9(4) COMP.
002900*
003000* Text for messages
003100*
003200 01  CW-MESSAGES.
003300   05 CW-MSG-PTR              PIC 9(4) COMP VALUE 1.
003400   05 CW-MSG-INVALID-FUNCTION PIC X(30)
003500      VALUE 'is an invalid function request'.
003510   05 CW-MSG-STORE-SUCCESSFUL  PIC X(20)
003520      VALUE ' stored successfully'.
003521   05 CW-MSG-DELETE-SUCCESSFUL PIC X(21)
003522      VALUE ' deleted successfully'.
003523   05 CW-MSG-NOTFOUND          PIC X(14)
003524      VALUE ' was not found'.
003530   05 CW-MSG-CICS-ERROR.
003550     10 FILLER          PIC X(22) VALUE 'Unexpected return code'.
003560     10 CW-MSG-EIBRESP  PIC 9(2).
003570     10 FILLER          PIC X(4)  VALUE ' on '.
003580     10 CW-MSG-FUNCTION PIC X(10).
003590     10 FILLER          PIC X(1)  VALUE ' '.
003591     10 CW-MSG-EIBRSRCE PIC X(8).
003600*
003700 LINKAGE SECTION.
003800 01  DFHCOMMAREA        PIC X(9000).
003900*
004000* LAYOUT FOR COMMMAREA
004100*
004200 01  CW-COMMAREA REDEFINES DFHCOMMAREA.
004300   05 CW-HEADER.
004400     10 CW-FUNCTION     PIC X(16).
004410     10 CW-FILE-NAME    PIC X(8).
004500     10 CW-KEY.
004501       15 CW-OBJID      PIC X(64).
004510       15 CW-SEQ        PIC 9(3).
004600     10 CW-MSGTXT       PIC X(80).
004610     10 CW-MIME-TYPE    PIC X(30).
004700     10 CW-SEG-COUNT    PIC 9(3).
004710     10 CW-OBJECT-SIZE  PIC 9(7).
004720     10 CW-DATA-LEN     PIC 9(5).
004730     10 CW-OBJDESC      PIC X(30).
004800   05 CW-DATA.
005000     10 CW-DATA-CONTENT PIC X(8000).
```

```
005100*
005200*  Record Layout for VSAM file used to store HTML Objects
005300*
005400 01  CWFILE-RECORD.
005410   05 CWFILE-HEADER.
005500     10 CWFILE-KEY.
005600       15 CWFILE-OBJID   PIC X(64).
005700       15 CWFILE-SEQ     PIC 9(03).
005800     10 CWFILE-NUM-RECS  PIC 9(03).
005900     10 CWFILE-MIME-TYPE PIC X(30).
005910     10 CWFILE-DATA-LEN  PIC 9(5).
005920     10 CWFILE-OBJECT-SIZE PIC 9(7).
005930     10 CWFILE-OBJDESC   PIC X(30).
006000   05 CWFILE-DATA.
006200     10 CWFILE-DATA-CONTENT PIC X(8000).
006300*
006400 EJECT.
006500 PROCEDURE DIVISION.
006600 A000-MAINLINE.
006700* Check that COMMAREA passed is long enough.
006800* If not, return immediately and do nothing.
006900     IF EIBCALEN < LENGTH OF CW-HEADER
007000        EXEC CICS RETURN END-EXEC
007100     END-IF.
007200*
007300     EVALUATE CW-FUNCTION
007400        WHEN 'Delete Object'    PERFORM B010-DELETE-OBJECT
007600        WHEN 'Retrieve Object'  PERFORM B020-RETRIEVE-OBJECT
007610        WHEN 'Store Object'     PERFORM B030-STORE-OBJECT
007700        WHEN OTHER PERFORM
007800             MOVE SPACES TO CW-MSGTXT
007900             STRING CW-FUNCTION DELIMITED BY SIZE
008000                    INTO CW-MSGTXT
008100                    WITH POINTER CW-MSG-PTR
008200             END-STRING
008300             STRING CW-MSG-INVALID-FUNCTION DELIMITED BY SIZE
008400                    INTO CW-MSGTXT
008500                    WITH POINTER CW-MSG-PTR
008600             END-STRING
008700             MOVE LOW-VALUE TO CW-DATA-CONTENT
008800             EXEC CICS RETURN END-EXEC
008900          END-PERFORM
009000     END-EVALUATE.
009100*
009200 A000-EXIT.
009300     GOBACK.
009400*
009500 B010-DELETE-OBJECT.
009510     EXEC CICS DELETE
009520               FILE(CW-FILE-NAME)
009530               RIDFLD(CW-OBJID)
009540               KEYLENGTH(LENGTH OF CW-OBJID)
009550               GENERIC
009560               NOHANDLE
009570     END-EXEC,
009580     IF EIBRESP = DFHRESP(NORMAL)
009581*        Build "successful completion" message and return to caller
009582         MOVE SPACES TO CW-MSGTXT
009583         STRING CW-OBJID DELIMITED BY SPACES
009584                INTO CW-MSGTXT
009585                WITH POINTER CW-MSG-PTR
009586         END-STRING
009587         STRING CW-MSG-DELETE-SUCCESSFUL DELIMITED BY SIZE
009588                INTO CW-MSGTXT
009589                WITH POINTER CW-MSG-PTR
009590         END-STRING
009592         EXEC CICS RETURN END-EXEC
009593     END-IF.
009594     IF EIBRESP = DFHRESP(NOTFND)
009595*        Build "Object not found" message and return to caller
009596         MOVE SPACES TO CW-MSGTXT
009597         STRING CW-OBJID DELIMITED BY SPACES
009598                INTO CW-MSGTXT
009599                WITH POINTER CW-MSG-PTR
009600         END-STRING
009601         STRING CW-MSG-NOTFOUND DELIMITED BY SIZE
009602                INTO CW-MSGTXT
009603                WITH POINTER CW-MSG-PTR
```

```
009604         END-STRING
009605         EXEC CICS RETURN END-EXEC
009606     END-IF.
009607*    Unexpected CICS response - build error message
009608     MOVE 'DELETE' TO CW-MSG-FUNCTION
009609     MOVE EIBRSRCE TO CW-MSG-EIBRSRCE
009610     MOVE EIBRESP  TO CW-MSG-EIBRESP
009611     MOVE CW-MSG-CICS-ERROR TO CW-MSGTXT
009612     MOVE LOW-VALUE TO CW-DATA-CONTENT
009613     EXEC CICS RETURN END-EXEC.
009910*
009920 B020-RETRIEVE-OBJECT.
009928     EXEC CICS READ
009929               FILE(CW-FILE-NAME)
009930               RIDFLD(CW-KEY)
009931               KEYLENGTH(LENGTH OF CWFILE-KEY)
009932               SET(ADDRESS OF CWFILE-RECORD)
009935               NOHANDLE
009936     END-EXEC,
009937     IF EIBRESP NOT = DFHRESP(NORMAL)
009938         MOVE 'READ' TO CW-MSG-FUNCTION
009939         MOVE EIBRSRCE TO CW-MSG-EIBRSRCE
009940         MOVE EIBRESP  TO CW-MSG-EIBRESP
009941         MOVE CW-MSG-CICS-ERROR TO CW-MSGTXT
009942         MOVE LOW-VALUE TO CW-DATA-CONTENT
009943         EXEC CICS RETURN END-EXEC
009944     END-IF.
009947     MOVE CWFILE-NUM-RECS     TO CW-SEG-COUNT.
009948     MOVE CWFILE-MIME-TYPE    TO CW-MIME-TYPE.
009949     MOVE CWFILE-OBJECT-SIZE  TO CW-OBJECT-SIZE.
009950     MOVE CWFILE-DATA-LEN     TO CW-DATA-LEN.
009951     MOVE CWFILE-DATA-CONTENT (1: CW-DATA-LEN) TO CW-DATA-CONTENT.
009960     EXEC CICS RETURN END-EXEC.
010000*
010100 B030-STORE-OBJECT.
010101* Delete any existing object with same name
010102     IF CW-SEQ = 1
010103         EXEC CICS DELETE
010104                   FILE(CW-FILE-NAME)
010105                   RIDFLD(CW-OBJID)
010106                   KEYLENGTH(LENGTH OF CW-OBJID)
010107                   GENERIC
010108                   NOHANDLE
010109         END-EXEC
010110     END-IF
010111     IF (EIBRESP NOT = DFHRESP(NORMAL) AND
010112         EIBRESP NOT = DFHRESP(NOTFND))
010113         MOVE 'DELETE' TO CW-MSG-FUNCTION
010114         MOVE EIBRSRCE TO CW-MSG-EIBRSRCE
010115         MOVE EIBRESP  TO CW-MSG-EIBRESP
010116         MOVE CW-MSG-CICS-ERROR TO CW-MSGTXT
010117         MOVE LOW-VALUE TO CW-DATA-CONTENT
010118         EXEC CICS RETURN END-EXEC
010119     END-IF.
010120* Build VSAM record and write to file
010121     EXEC CICS GETMAIN
010122        SET(ADDRESS OF CWFILE-RECORD)
010123        LENGTH(LENGTH OF CWFILE-RECORD)
010124     END-EXEC.
010125     MOVE CW-OBJID        TO CWFILE-OBJID.
010126     MOVE CW-SEQ          TO CWFILE-SEQ.
010127     MOVE CW-SEG-COUNT    TO CWFILE-NUM-RECS.
010128     MOVE CW-MIME-TYPE    TO CWFILE-MIME-TYPE.
010129     MOVE CW-DATA-LEN     TO CWFILE-DATA-LEN.
010130     MOVE CW-OBJECT-SIZE  TO CWFILE-OBJECT-SIZE.
010131     MOVE CW-OBJDESC      TO CWFILE-OBJDESC.
010132     MOVE CW-DATA-CONTENT TO CWFILE-DATA-CONTENT.
010133     COMPUTE CW-RECLEN = CWFILE-DATA-LEN + LENGTH OF CWFILE-HEADER.
010134     EXEC CICS WRITE
010135               FILE(CW-FILE-NAME)
010136               RIDFLD(CWFILE-KEY)
010137               KEYLENGTH(LENGTH OF CWFILE-KEY)
010138               FROM(CWFILE-RECORD)
010139               LENGTH(CW-RECLEN)
010140               NOHANDLE
010141     END-EXEC,
010142     IF EIBRESP NOT = DFHRESP(NORMAL)
010143         MOVE 'WRITE' TO CW-MSG-FUNCTION
```

```
010144         MOVE EIBRSRCE TO CW-MSG-EIBRSRCE
010145         MOVE EIBRESP  TO CW-MSG-EIBRESP
010146         MOVE CW-MSG-CICS-ERROR TO CW-MSGTXT
010147         MOVE LOW-VALUE TO CW-DATA-CONTENT
010148         EXEC CICS RETURN END-EXEC
010149     END-IF.
010150* Build "successful completion" message and return to caller
010151     MOVE SPACES TO CW-MSGTXT.
010152     STRING CW-OBJID DELIMITED BY SPACES
010153            INTO CW-MSGTXT
010154            WITH POINTER CW-MSG-PTR
010155     END-STRING.
010160     STRING CW-MSG-STORE-SUCCESSFUL DELIMITED BY SIZE
010170            INTO CW-MSGTXT
010180            WITH POINTER CW-MSG-PTR
010190     END-STRING.
010191     MOVE LOW-VALUE TO CW-DATA-CONTENT.
010200     EXEC CICS RETURN END-EXEC.
010300*
```

B.6 CICS COBOL Program to List Objects

```
000100*****************************************************************
000200*    CWLIST - CICS Server program for Web Sample                *
000300*****************************************************************
000400*    CREATED 95/08/10 BY David Thrum                            *
000500*                                                               *
000600*    COPYRIGHT - IBM                                            *
000700*                                                               *
000800*    FUNCTION  - Called from Web Server via ECI                 *
000900*              - Lists objects stored in CICS "Object Store"    *
001500*    PARAMETERS PASSED                                          *
001600*              - CICS COMMAREA                                  *
001700*                                                               *
001710*    This is an example of a program that is "HTML-aware".      *
001720*    It formats the data returned as HTML that can be passed    *
001730*    directly to the Web Browser for display.                   *
001770*                                                               *
001800*****************************************************************
001900 IDENTIFICATION DIVISION.
002000 PROGRAM-ID. CWLIST.
002100 ENVIRONMENT DIVISION.
002200 DATA DIVISION.
002300*
002400 WORKING-STORAGE SECTION.
002500*
002600*  MISCELLANEOUS WORK AREAS/CONSTANTS
002700*
002840 77  WS-KEY-GT-FLAG      PIC X(1) VALUE 'N'.
002841 77  WS-KEY              PIC X(67).
002842 77  WS-KEYLEN           PIC S9(4) COMP.
002850 77  CW-TXT-PTR          PIC 9(5) COMP VALUE 1.
002860 77  WS-PREPEND-STRING   PIC X(64) VALUE SPACES.
002900*
003000* Text for messages
003100*
003200 01  CW-MESSAGES.
003300   05 CW-MSG-PTR              PIC 9(4) COMP VALUE 1.
003400   05 CW-MSG-INVALID-FUNCTION PIC X(30)
003500      VALUE 'is an invalid function request'.
003523   05 CW-MSG-NOTFOUND          PIC X(14)
003524      VALUE ' was not found'.
003530   05 CW-MSG-CICS-ERROR.
003550     10 FILLER          PIC X(22) VALUE 'Unexpected return code'.
003560     10 CW-MSG-EIBRESP  PIC 9(2).
003570     10 FILLER          PIC X(4)  VALUE ' on '.
003580     10 CW-MSG-FUNCTION PIC X(10).
003590     10 FILLER          PIC X(1)  VALUE ' '.
003591     10 CW-MSG-EIBRSRCE PIC X(8).
003600*
003700 LINKAGE SECTION.
003800 01  DFHCOMMAREA        PIC X(32500).
003900*
004000* LAYOUT FOR COMMMAREA
004100*
004200 01  CW-COMMAREA REDEFINES DFHCOMMAREA.
004300   05 CW-HEADER.
004400     10 CW-FUNCTION     PIC X(16).
004410     10 CW-FILE-NAME    PIC X(8).
004500     10 CW-KEY.
004501       15 CW-OBJID      PIC X(64).
004510       15 CW-SEQ        PIC 9(3).
004601     10 CW-MSGTXT       PIC X(80).
004610     10 CW-MIME-TYPE    PIC X(30).
004620     10 CW-KEYLEN       PIC 9(02).
004800   05 CW-DATA.
005000     10 CW-DATA-LENGTH  PIC 9(5).
005010     10 CW-DATA-CONTENT PIC X(32000).
005100*
005200*  Record Layout for VSAM file used to store HTML Objects
005300*
005400 01  CWFILE-RECORD.
005410   05 CWFILE-HEADER.
005500     10 CWFILE-KEY.
005600       15 CWFILE-OBJID   PIC X(64).
005700       15 CWFILE-SEQ     PIC 9(03).
005800     10 CWFILE-NUM-RECS  PIC 9(03).
```

```
005900     10 CWFILE-MIME-TYPE PIC X(30).
005910     10 CWFILE-DATA-LEN  PIC 9(5).
005920     10 CWFILE-OBJECT-SIZE PIC 9(7).
005930     10 CWFILE-OBJDESC   PIC X(30).
006000   05 CWFILE-DATA.
006200     10 CWFILE-DATA-CONTENT PIC X(8000).
006300*
006400 EJECT.
006500 PROCEDURE DIVISION.
006600 A000-MAINLINE.
006700* Check that COMMAREA passed is long enough.
006800* If not, return immediately and do nothing.
006900     IF EIBCALEN < LENGTH OF CW-HEADER
007000        EXEC CICS RETURN END-EXEC
007100     END-IF.
007110*
007200* Check that we are called with correct function request
007300     EVALUATE CW-FUNCTION
007400        WHEN 'List Objects'  PERFORM B010-LIST-OBJECTS
007700        WHEN OTHER PERFORM
007800              MOVE SPACES TO CW-MSGTXT
007900              STRING CW-FUNCTION DELIMITED BY SIZE
008000                     INTO CW-MSGTXT
008100                     WITH POINTER CW-MSG-PTR
008200              END-STRING
008300              STRING CW-MSG-INVALID-FUNCTION DELIMITED BY SIZE
008400                     INTO CW-MSGTXT
008500                     WITH POINTER CW-MSG-PTR
008600              END-STRING
008700              MOVE LOW-VALUE TO CW-DATA-CONTENT
008800              EXEC CICS RETURN END-EXEC
008900          END-PERFORM
009000     END-EVALUATE.
009100*
009200 A000-EXIT.
009300     GOBACK.
009400*
009500 B010-LIST-OBJECTS.
009600* get string to prepend to file key to build URL in anchor
009620     MOVE CW-DATA-CONTENT (1: CW-DATA-LENGTH) TO WS-PREPEND-STRING
009700     IF CW-KEYLEN IS NUMERIC MOVE CW-KEYLEN TO WS-KEYLEN
009800     ELSE MOVE 0 TO WS-KEYLEN.
009900     IF WS-KEYLEN = 0 THEN
009910         MOVE 1 TO WS-KEYLEN
009920         MOVE 0 TO CW-KEYLEN
009928         EXEC CICS STARTBR
009929                   FILE(CW-FILE-NAME)
009930                   RIDFLD(CW-KEY)
009931                   KEYLENGTH(WS-KEYLEN)
009932                   GTEQ
009933                   GENERIC
009935                   NOHANDLE
009936         END-EXEC
009937     ELSE
009938         MOVE CW-KEY TO WS-KEY
009939         EXEC CICS STARTBR
009940                   FILE(CW-FILE-NAME)
009941                   RIDFLD(CW-KEY)
009942                   KEYLENGTH(WS-KEYLEN)
009943                   EQUAL
009944                   GENERIC
009945                   NOHANDLE
009946         END-EXEC
009947     END-IF.
009948     IF EIBRESP = DFHRESP(NOTFND)
009949*        Build "Object not found" message and return to caller
009950         MOVE SPACES TO CW-MSGTXT
009951         STRING CW-OBJID DELIMITED BY SPACES
009952                INTO CW-MSGTXT
009953                WITH POINTER CW-MSG-PTR
009954         END-STRING
009955         STRING CW-MSG-NOTFOUND DELIMITED BY SIZE
009956                INTO CW-MSGTXT
009957                WITH POINTER CW-MSG-PTR
009958         END-STRING
009959         EXEC CICS RETURN END-EXEC
009960     END-IF.
009961*    Unexpected CICS response - build error message
```

```
009962     IF EIBRESP NOT = DFHRESP(NORMAL)
009963         MOVE 'STARTBR' TO CW-MSG-FUNCTION
009964         MOVE EIBRSRCE TO CW-MSG-EIBRSRCE
009965         MOVE EIBRESP TO CW-MSG-EIBRESP
009966         MOVE CW-MSG-CICS-ERROR TO CW-MSGTXT
009967         MOVE LOW-VALUE TO CW-DATA-CONTENT
009968         EXEC CICS RETURN END-EXEC
009969     END-IF.
009970*
009985     STRING '<TABLE BORDER=4><THEAD><TR><TH align=left><BR>'
009986            DELIMITED BY SIZE
009987            INTO CW-DATA-CONTENT
009988            WITH POINTER CW-TXT-PTR
009989     END-STRING.
009990     STRING 'Object Name<TH align=left><br>Description<TH>'
009991            DELIMITED BY SIZE
009992            INTO CW-DATA-CONTENT
009993            WITH POINTER CW-TXT-PTR
009994     END-STRING.
009995     STRING 'Content<br>Type<TH>Object<br>Size</thead><BODY>'
009996            DELIMITED BY SIZE
009997            INTO CW-DATA-CONTENT
009998            WITH POINTER CW-TXT-PTR
009999     END-STRING.
010000*
010001*    Initial read..
010002*
010003     EXEC CICS READNEXT
010004               FILE(CW-FILE-NAME)
010005               RIDFLD(CW-KEY)
010006               KEYLENGTH(WS-KEYLEN)
010007               SET(ADDRESS OF CWFILE-RECORD)
010008               NOHANDLE
010009     END-EXEC,
010010*
010011* Read all records starting from supplied partial key
010012*
010013     PERFORM UNTIL EIBRESP NOT = DFHRESP(NORMAL) OR
010014                   WS-KEY-GT-FLAG = 'Y'
010015         IF EIBRESP NOT = DFHRESP(NORMAL)
010016             MOVE 'READ' TO CW-MSG-FUNCTION
010017             MOVE EIBRSRCE TO CW-MSG-EIBRSRCE
010018             MOVE EIBRESP TO CW-MSG-EIBRESP
010019             MOVE CW-MSG-CICS-ERROR TO CW-MSGTXT
010020             MOVE LOW-VALUE TO CW-DATA-CONTENT
010021             EXEC CICS RETURN END-EXEC
010022         END-IF
010023         IF CWFILE-SEQ = 1
010024             STRING '<TR><TD><A HREF="' DELIMITED BY SIZE
010025                    INTO CW-DATA-CONTENT
010026                    WITH POINTER CW-TXT-PTR
010027             END-STRING
010028             STRING WS-PREPEND-STRING DELIMITED BY SPACES
010029                    INTO CW-DATA-CONTENT
010030                    WITH POINTER CW-TXT-PTR
010031             END-STRING
010032             STRING CW-KEY DELIMITED BY SPACES
010033                    INTO CW-DATA-CONTENT
010034                    WITH POINTER CW-TXT-PTR
010035             END-STRING
010036             STRING '">' DELIMITED BY SIZE
010037                    INTO CW-DATA-CONTENT
010038                    WITH POINTER CW-TXT-PTR
010039             END-STRING
010040             STRING CW-KEY DELIMITED BY SPACES
010041                    INTO CW-DATA-CONTENT
010042                    WITH POINTER CW-TXT-PTR
010043             END-STRING
010044             STRING '</A><TD>' DELIMITED BY SIZE
010045                    INTO CW-DATA-CONTENT
010046                    WITH POINTER CW-TXT-PTR
010047             END-STRING
010048             STRING CWFILE-OBJDESC DELIMITED BY SIZE
010049                    INTO CW-DATA-CONTENT
010050                    WITH POINTER CW-TXT-PTR
010051             END-STRING
010052             STRING '<TD>' DELIMITED BY SIZE
010053                    INTO CW-DATA-CONTENT
```

```
010054                  WITH POINTER CW-TXT-PTR
010055             END-STRING
010056             STRING CWFILE-MIME-TYPE DELIMITED BY SPACES
010057                    INTO CW-DATA-CONTENT
010058                    WITH POINTER CW-TXT-PTR
010059             END-STRING
010060             STRING '<TD>' DELIMITED BY SIZE
010061                    INTO CW-DATA-CONTENT
010062                    WITH POINTER CW-TXT-PTR
010063             END-STRING
010064             STRING CWFILE-OBJECT-SIZE DELIMITED BY SPACES
010065                    INTO CW-DATA-CONTENT
010066                    WITH POINTER CW-TXT-PTR
010067             END-STRING
010068         END-IF
010069         EXEC CICS READNEXT
010070                   FILE(CW-FILE-NAME)
010071                   RIDFLD(CW-KEY)
010072                   KEYLENGTH(WS-KEYLEN)
010073                   SET(ADDRESS OF CWFILE-RECORD)
010074                   NOHANDLE
010075         END-EXEC
010076*        Check if we have passed value of generic key
010077         IF CW-KEYLEN NOT = 0 AND
010078             CW-KEY (1: WS-KEYLEN) NOT = WS-KEY (1: WS-KEYLEN)
010079             MOVE 'Y' TO WS-KEY-GT-FLAG
010080         END-IF
010081     END-PERFORM
010082*
010083* Finish formatting list display
010084*
010085     STRING '</TABLE><P>' DELIMITED BY SIZE
010086            INTO CW-DATA-CONTENT
010087            WITH POINTER CW-TXT-PTR
010088     END-STRING.
010089     EXEC CICS ENDBR
010090               FILE(CW-FILE-NAME)
010091               NOHANDLE
010092     END-EXEC.
010093     COMPUTE CW-DATA-LENGTH = CW-TXT-PTR - 1.
010094     EXEC CICS RETURN END-EXEC.
010100*
010300*
```

B.7 CICS Data Conversion Table

```
*
*
         DFHCNV TYPE=INITIAL,CDEPAGE=(437)
*        DFHCNV TYPE=INITIAL,CDEPAGE=(437,USRD)
*
* THIS TABLE IS NEEDED ON THE HOST FOR ASCII <-> EBCDIC DATA CONVERSION
*
*         :::           :::            :::            :::
*
         DFHCNV TYPE=ENTRY,RTYPE=PC,RNAME=CWLIST,USREXIT=NO
         DFHCNV TYPE=SELECT,OPTION=DEFAULT
         DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=32767,   X
               LAST=YES
*
         DFHCNV TYPE=ENTRY,RTYPE=PC,RNAME=CWSERVER,USREXIT=NO
         DFHCNV TYPE=SELECT,OPTION=DEFAULT
         DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=246,     X
               LAST=YES
*
*         :::           :::            :::            :::
*
LABLN    DFHCNV TYPE=FINAL
         END   DFHCNVBA
```

B.8 VSAM File Definition

```
//CICSRS1V JOB (999,POK),'CICSRS1',NOTIFY=CICSRS1,
//    CLASS=A,MSGCLASS=T,TIME=1439,
//    MSGLEVEL=(1,1)
//*
//* DELETE/DEFINE VSAM KSDS USED BY CICSWEB SAMPLE APPLICATION
//*
//DEFINE EXEC PGM=IDCAMS,REGION=1M
//AMSDUMP  DD   SYSOUT=*
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
     DELETE (CICSRS1.CWFILE) PURGE CLUSTER
     DEFINE CLUSTER (NAME(CICSRS1.CWFILE)    -
                     RECSZ(256 8184)            -
                     REC(100 100)               -
                     KEYS(67,0)                 -
                     INDEXED                    -
                     UNIQUE                     -
                     IMBED                      -
                     FREESPACE(0 20)            -
                     VOLUMES(STCIC1))           -
            DATA    (NAME(CICSRS1.CWFILE.DATA)   -
                     CISZ(8192))                -
            INDEX   (NAME(CICSRS1.CWFILE.INDEX))
     /* LOAD A DUMMY RECORD */
     REPRO INFILE(DATAIN) OUTDATASET(CICSRS1.CWFILE)
//*
//DATAIN DD *
   INITIAL RECORD------------------------------
//*
```

Appendix C. CICS/ESA State Management Sample

This appendix contains the source code for the sample CICS/ESA state management program CICSSTAT and the sample application that illustrates the use of CICSSTAT. It includes the following components:

- REXX CGI script.
- COBOL CICS/ESA Web server application program.
- Assembler CICS/ESA state management program.

C.1 REXX CGI Script

```
/*********************************************************************/
/* A Forms Program which will use state information to identify the */
/* source of the request, and retrieve any stored data associated   */
/* with that user from the form.                                     */
/*********************************************************************/

say "Content-type: text/html"
say""

/*      If there is any forms data for us to read    */
/*      read it in and parse it.                     */

env = 'OS2ENVIRONMENT'
cl = value('CONTENT_LENGTH',,env)

if cl > 0 then do
  newitem = ""
 len = c2d(cl)

 data = charin(,,len)
   /* Set input fields from the incoming data stream  */
  data=data'&'
  do until data=''
    parse var data assign '&' data    /* Parse data for input fields      */
    parse var assign inname '=' value /* 'name=' to assign, value to value*/
    value = translate(value,' ','2b090a0d'x) /* Get rid of '+', CR, LF    */
    inname = packur(inname)           /* call packur to decode esc chars  */

    /*  Put the input parameters into corresponding variables */
    select
     when inname = "handle"  then
        handle = value
     when inname = "function" then
        function = packur(value)
    when inname = "action" then
        action = packur(value)
    when inname = "state" then
        state = packur(value)
     when inname = "submit-b" then
        submitb = packur(value)
                when inname = "name" then
        name = packur(value)
      when inname = "address" then
       address = packur(value)
      otherwise
      do
       newitem = packur(value)
      end
    end /* select */
 end /* parsing of input data */

 /*  Now build parameter list for CICS */
 if handle = "" then
  /* a logic error has occurred since if there is forms output, */
  /* we should always have a handle                             */
  /* Issue an error message and start again                     */
  do
   say "<P>A logic error has occurred. Please re-register with CICS"
   commarea = left("IDENTIFY",1024,' ')
```

```
  end   /* no handle in form */

 else
  do
    commarea = left(function,10,' ')
    commarea = commarea||handle
    commarea = commarea||left(action,6.' ')
    commarea = commarea||left(name,30,' ')
    commarea = commarea||left(address,70,' ')
    commarea = commarea||left(newitem,390,' ')
  end
end

else
 /*************************************************************/
 /* There is no forms data to be read in, so this must be a new  */
 /* conversation. Build the appropriate parameter               */
 /* list and pass it to CICS.                                   */
 /*************************************************************/
 do
  commarea = "IDENTIFY  "
end
say ecicall(commarea)
return

ecicall: procedure
arg incomm

/*********************************************************************/
/*                                                                   */
/*  This program demonstrates a synchronous, non-extended call to  */
/*  CICS program (SAMPLE) using RxECI with DEBUG enabled.           */
/*                                                                   */
/*********************************************************************/
  If rxfuncquery('ECI') Then Do
     If rxfuncadd('ECI', 'RXECI', 'RxECI' ) Then signal failure
   /*  say "ECI registered" */
  End
DEBUG=0
DEBUGFILE="rxeci.log"

/* provide a simple COMMAREA just filled with '.' */
  commarea = left(incomm,1024,'00'x)
/* default is "ECIPROG " */
  ECI.userid            = "SYSAD"
  ECI.program_name      = "SWLIST2"

  commarea2 =ECI('ECI.',commarea)
  rc = check_rc(commarea2)

/* throw the following line away in non-development mode */
  rc = rxfuncdrop('ECI')

return commarea2


/*************************************************************/
/*           some utility functions                          */
/*************************************************************/

check_rc: procedure  expose ECI.
arg commarea

  msgfile = "faaeci.msg"
  rc = show_eci_stem()
  if (ECI.return_code \= 0) then do
     say "RxECI failed"
     do until lines(msgfile) = 0
        line = linein(msgfile)
        parse var line returncode message
        if ECI.return_code = returncode then say message
        end
     end
     rc = lineout(msgfile)
return(0)

show_eci_stem: procedure  expose ECI.
```

```
return(0)

/* return code from ECI */
  ECI.return_code       = ""                     /* response  */

FAILURE:
  Say 'Failure to add function at line' sigl
  Address CMD "PAUSE"
  Return 1
```

C.2 COBOL CICS/ESA Web Server Application Program

```
      ****************************************************************
      *    SWLIST2- CICS Server program for Web State Sample         *
      ****************************************************************
      *    CREATED 95/08/10 BY Steve Wall                            *
      *                                                              *
      *    COPYRIGHT - IBM                                           *
      *                                                              *
      *    FUNCTION  - Called from Web Server via ECI                *
      *              - Builds lists of objects on TS in CICS         *
      *    PARAMETERS PASSED                                         *
      *              - CICS COMMAREA                                 *
      *                                                              *
      *    This is an example of a program that is "HTML-aware".     *
      *    It formats the data returned as HTML that can be passed   *
      *    directly to the Web Browser for display.                  *
      *                                                              *
      ****************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SWLIST2.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
      *
       WORKING-STORAGE SECTION.
      *
      *  MISCELLANEOUS WORK AREAS/CONSTANTS
      *
      *
      *    FUNCTION CONSTANTS
      *
       77 SW-IDENTIFY PIC X(10) VALUE 'IDENTIFY  '.
       77 SW-ADD      PIC X(10) VALUE 'ADD       '.
       77 SW-DELETE   PIC X(10) VALUE 'DELETE    '.
       77 SW-REGISTER PIC X(10) VALUE 'REGISTER  '.
      *
      *    STATE COMMAREA CONSTANTS
      *
       77 ST-CREATE   PIC X(1) VALUE 'C'.
       77 ST-RETRIEVE PIC X(1) VALUE 'R'.
       77 ST-STORE    PIC X(1) VALUE 'S'.
       77 ST-DESTROY  PIC X(1) VALUE 'D'.
       77 ST-EYE-INIT PIC X(4) VALUE 'COM>'.
      *
       77 STATE-MANAGER PIC X(8) VALUE 'CICSSTAT'.
       77 QUEUE-TEXT    PIC X(80).
       77 TSQ-LENGTH-VALUE  PIC 9(4) COMP VALUE 80.
      *
      *      STATE COMMAREA
      *
       01 ST-COMMAREA.
        05 ST-EYECATCHER  PIC X(4).
        05 ST-FUNCTION    PIC X(1).
        05 ST-RETCODE     PIC X(1).
        05 FILLER         PIC X(2).
        05 ST-HANDLE      PIC 9(8) COMP.
        05 ST-DATA.
         10 ST-NAME           PIC X(30).
         10 ST-ADDRESS        PIC X(70).
         10 ST-FORM-FUNCTION  PIC X(1).
         10 FILLER            PIC X(3).
         10 ST-TSQ-NAME.
          15 ST-TSQ-FIRST     PIC X(4).
          15 ST-TSQ-SECOND    PIC 9(8) COMP.
```

```
   10 ST-ITEMNUM        PIC S9(4) COMP.
   10 FILLER        PIC X(133).
*
* Text for messages
*
 01  MSG-TXT            PIC X(80).
 01  TEMPNUM            PIC S9(4) COMP.
 01  ITEMNUM            PIC S9(4) COMP.
 01  CHAR-HANDLE        PIC 9(8).
 01  SW-TXT-PTR          PIC 9(4) COMP VALUE 1.
 01  ST-COMMAREA-LENGTH PIC 9(9) COMP VALUE 268.
 01  TSQ-LENGTH         PIC 9(4) COMP.
*
 LINKAGE SECTION.
 01  DFHCOMMAREA        PIC X(500).
*
* LAYOUT FOR COMMAREA
*
 01  SW-COMMAREA REDEFINES DFHCOMMAREA.
   05 SW-HEADER.
     10 SW-FUNCTION     PIC X(10).
     10 SW-HANDLE       PIC 9(8).
     10 SW-ACTION       PIC X(6).
     10 SW-NAME         PIC X(30).
     10 SW-ADDRESS      PIC X(70).
     10 SW-SPARE        PIC X(900).
*
 01  SW-OUTPUT REDEFINES SW-COMMAREA.
  05 SW-DATA-CONTENT    PIC X(1024).
*
 EJECT.
 PROCEDURE DIVISION.
 A000-MAINLINE.
* Check that COMMAREA passed is long enough.
* If not, return immediately and do nothing.
     IF EIBCALEN < LENGTH OF SW-HEADER
        EXEC CICS RETURN END-EXEC
     END-IF.
*
*    Save name and address in local storage
*
     EVALUATE SW-FUNCTION
     WHEN 'IDENTIFY  ' PERFORM B010-IDENTIFY
                       THRU    B010-IDENTIFY-EXIT
     WHEN 'REGISTER  ' PERFORM C020-REGISTER-USER
                       THRU    C020-REGISTER-USER-EXIT
     WHEN 'ADD       ' PERFORM H070-SELECT-ACTION
                       THRU    H070-SELECT-ACTION-EXIT
     WHEN 'DELETE    ' PERFORM E040-DELETE-LIST
                       THRU    E040-DELETE-LIST-EXIT
     WHEN OTHER        PERFORM F050-INVALID-FUNCTION
                       THRU    F050-INVALID-FUNCTION-EXIT
     END-EVALUATE.

     EXEC CICS RETURN END-EXEC.
*
 A000-EXIT.
     GOBACK.
*
 B010-IDENTIFY.
*****************************************************************
*    Create a new status block for this new conversation        *
*****************************************************************
     MOVE  ST-CREATE TO ST-FUNCTION.
     MOVE  ST-EYE-INIT TO ST-EYECATCHER.
     EXEC CICS LINK PROGRAM('CICSSTAT')
               COMMAREA(ST-COMMAREA)
               LENGTH(ST-COMMAREA-LENGTH)
               END-EXEC.
     IF EIBRESP NOT EQUAL DFHRESP(NORMAL) THEN
*****************************************************************
*    If the link was successful and the State Manager returned
*    no error, go ahead and update our list
*****************************************************************
       MOVE 'Link to State Manager failed   ' TO MSG-TXT
       PERFORM J070-ERROR THRU J070-ERROR-EXIT.
     IF ST-RETCODE NOT EQUAL LOW-VALUE THEN
       MOVE 'State Manager detected error   ' TO MSG-TXT
```

```
       PERFORM J070-ERROR THRU J070-ERROR-EXIT.
     MOVE  ST-HANDLE TO CHAR-HANDLE
     MOVE  SW-REGISTER TO ST-FORM-FUNCTION.
     PERFORM B010-IDENTIFY-BUILD THRU B010-IDENTIFY-BUILD-EXIT.
 B010-IDENTIFY-EXIT.
*
 C020-REGISTER-USER.
     MOVE ST-RETRIEVE TO ST-FUNCTION.
     MOVE SW-HANDLE TO ST-HANDLE.
     EXEC CICS LINK PROGRAM(STATE-MANAGER)
                    COMMAREA(ST-COMMAREA)
                    LENGTH(ST-COMMAREA-LENGTH)
                    NOHANDLE
                    END-EXEC.
     MOVE   SW-NAME TO ST-NAME.
     MOVE   SW-ADDRESS TO ST-ADDRESS.
     MOVE   SW-ADD TO ST-FORM-FUNCTION.
     STRING SW-NAME
            DELIMITED BY SIZE
            INTO ST-TSQ-FIRST
     END-STRING.
     MOVE   ST-HANDLE TO ST-TSQ-SECOND.
     MOVE ST-STORE TO ST-FUNCTION.
     MOVE SW-HANDLE TO ST-HANDLE.
     EXEC CICS LINK PROGRAM(STATE-MANAGER)
                    COMMAREA(ST-COMMAREA)
                    LENGTH(ST-COMMAREA-LENGTH)
                    NOHANDLE
                    END-EXEC.
     IF EIBRESP NOT EQUAL DFHRESP(NORMAL) THEN
*********************************************************************
*    If the link was successful and the State Manager returned
*    no error, go ahead and update our list
*********************************************************************
       MOVE 'Link to State Manager failed   ' TO MSG-TXT
       PERFORM J070-ERROR THRU J070-ERROR-EXIT.
     IF ST-RETCODE NOT EQUAL LOW-VALUE THEN
       MOVE 'State Manager detected error   ' TO MSG-TXT
       PERFORM J070-ERROR THRU J070-ERROR-EXIT.
     PERFORM C020-REGISTER-BUILD THRU C020-REGISTER-BUILD-EXIT.
 C020-REGISTER-USER-EXIT.
 H070-SELECT-ACTION.
     EVALUATE SW-ACTION
       WHEN 'ADD   '
         PERFORM D030-ADD-ITEM THRU D030-ADD-ITEM-EXIT
       WHEN 'DELETE'
         PERFORM E040-DELETE-LIST THRU E040-DELETE-LIST-EXIT
       WHEN OTHER
         PERFORM D030-ADD-ITEM THRU D030-ADD-ITEM-EXIT
     END-EVALUATE.
 H070-SELECT-ACTION-EXIT.
 D030-ADD-ITEM.
     MOVE ST-RETRIEVE TO ST-FUNCTION.
     MOVE SW-HANDLE TO CHAR-HANDLE
     MOVE SW-HANDLE TO ST-HANDLE.
*********************************************************************
*    Link to the State Manager to retrieve our state data          *
*********************************************************************
     EXEC CICS LINK PROGRAM(STATE-MANAGER)
                    COMMAREA(ST-COMMAREA)
                    LENGTH(ST-COMMAREA-LENGTH)
                    NOHANDLE
                    END-EXEC
     IF EIBRESP NOT EQUAL DFHRESP(NORMAL) THEN
*********************************************************************
*    If the link was successful and the State Manager returned
*    no error, go ahead and update our list
*********************************************************************
       MOVE 'Link to State Manager failed   ' TO MSG-TXT
       PERFORM J070-ERROR THRU J070-ERROR-EXIT.
     IF ST-RETCODE NOT EQUAL LOW-VALUE THEN
       MOVE 'State Manager detected error   ' TO MSG-TXT
       PERFORM J070-ERROR THRU J070-ERROR-EXIT.
*********************************************************************
*     If name and address in the form do not match state data then *
*     raise an error                                               *
*********************************************************************
      IF  ST-NAME NOT = SW-NAME THEN
```

```
           MOVE 'STATE MISMATCH IN NAME FIELD   ' TO MSG-TXT
           PERFORM J070-ERROR THRU J070-ERROR-EXIT.
      IF  ST-ADDRESS NOT = SW-ADDRESS THEN
           MOVE 'STATE MISMATCH IN ADDRESS FIELD' TO MSG-TXT
           PERFORM J070-ERROR THRU J070-ERROR-EXIT.
*****************************************************************
*     Write the new data to the TS Queue                        *
*****************************************************************
      MOVE ST-ITEMNUM TO ITEMNUM.
      COMPUTE ITEMNUM = ITEMNUM + 1.
      STRING SW-SPARE
            DELIMITED BY SPACES
            INTO QUEUE-TEXT
      END-STRING.
      EXEC CICS WRITEQ TS
                QUEUE(ST-TSQ-NAME)
                ITEM(ITEMNUM)
                FROM(QUEUE-TEXT)
                LENGTH(80)
                NOHANDLE
                END-EXEC
      IF EIBRESP = DFHRESP(NORMAL) THEN
*****************************************************************
*      If the write was OK, update our state data               *
*****************************************************************
       MOVE ITEMNUM TO ST-ITEMNUM
       MOVE ST-STORE TO ST-FUNCTION
       EXEC CICS LINK PROGRAM(STATE-MANAGER)
                      COMMAREA(ST-COMMAREA)
                      LENGTH(ST-COMMAREA-LENGTH)
                      NOHANDLE
                      END-EXEC
       PERFORM D030-ADD-BUILD THRU D030-ADD-BUILD-EXIT
      END-IF.
 D030-ADD-ITEM-EXIT.
 E040-DELETE-LIST.
     MOVE ST-RETRIEVE TO ST-FUNCTION.
     MOVE SW-HANDLE TO CHAR-HANDLE
     MOVE SW-HANDLE TO ST-HANDLE.
*****************************************************************
*    Link to the State Manager to retrieve our state data       *
*****************************************************************
       EXEC CICS LINK PROGRAM(STATE-MANAGER)
                      COMMAREA(ST-COMMAREA)
                      LENGTH(ST-COMMAREA-LENGTH)
                      NOHANDLE
                      END-EXEC
     IF EIBRESP NOT EQUAL DFHRESP(NORMAL) THEN
*****************************************************************
*    If the link was successful and the State Manager returned
*    no error, go ahead and update our list
*****************************************************************
       MOVE 'Link to State Manager failed   ' TO MSG-TXT
       PERFORM J070-ERROR THRU J070-ERROR-EXIT.
     IF ST-RETCODE NOT EQUAL LOW-VALUE THEN
       MOVE 'State Manager detected error   ' TO MSG-TXT
       PERFORM J070-ERROR THRU J070-ERROR-EXIT.
*****************************************************************
*     If name and address in the form do not match state data then *
*     raise an error                                            *
*****************************************************************
     IF  ST-NAME NOT EQUAL SW-NAME THEN
       MOVE 'Invalid name supplied in form  ' TO MSG-TXT
       PERFORM J070-ERROR THRU J070-ERROR-EXIT.
     IF  ST-ADDRESS NOT EQUAL SW-ADDRESS THEN
       MOVE 'Invalid address supplied in form' TO MSG-TXT
       PERFORM J070-ERROR THRU J070-ERROR-EXIT.
*****************************************************************
*     Delete our TS queue                                       *
*****************************************************************
      EXEC CICS DELETEQ TS
                QUEUE(ST-TSQ-NAME)
                NOHANDLE
                END-EXEC
      IF EIBRESP = DFHRESP(NORMAL) THEN
*****************************************************************
*      If the write was OK, update our state data               *
*****************************************************************
```

```
       MOVE ITEMNUM TO ST-ITEMNUM.
       MOVE ST-DESTROY TO ST-FUNCTION.
       EXEC CICS LINK PROGRAM(STATE-MANAGER)
                      COMMAREA(ST-COMMAREA)
                      LENGTH(ST-COMMAREA-LENGTH)
                      NOHANDLE
                      END-EXEC
     IF EIBRESP NOT EQUAL DFHRESP(NORMAL) THEN
*********************************************************************
*    If the link was successful and the State Manager returned
*    no error, go ahead and update our list
*********************************************************************
       MOVE 'Link to State Manager failed   ' TO MSG-TXT
       PERFORM J070-ERROR THRU J070-ERROR-EXIT.
     IF ST-RETCODE NOT EQUAL LOW-VALUE THEN
       MOVE 'State Manager detected error   ' TO MSG-TXT
       PERFORM J070-ERROR THRU J070-ERROR-EXIT.
     PERFORM E040-DELETE-BUILD THRU E040-DELETE-BUILD-EXIT.
 E040-DELETE-LIST-EXIT.
 B010-IDENTIFY-BUILD.
     MOVE LOW-VALUE TO SW-DATA-CONTENT.
     STRING '<HTML><HEAD><TITLE>NAME AND ADDRESS</TITLE></HEAD>'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING '<IMG SRC="http://ladoga.sanjose.ibm.com/'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING 'slwhtml/slwimg/cwmast.gif" ALT="CICSWEB Masthead">'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING '<P>This is a sample application written to ill'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING 'ustrate how the CICSSTAT sample application ca'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING 'n be used to manage state information.        '
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING 'It creates, adds to, and deletes, a shopping li'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING 'st held on CICS Temporary Storage.            '
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING '<FORM METHOD=POST ACTION="/cgi-bin/REXXCIC3" >'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING '<INPUT TYPE="HIDDEN" NAME="function" '
              DELIMITED BY SIZE
              INTO SW-DATA-CONTENT
              WITH POINTER SW-TXT-PTR
     STRING 'VALUE="REGISTER">'
              DELIMITED BY SIZE
              INTO SW-DATA-CONTENT
              WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING '<INPUT TYPE="HIDDEN" NAME="handle" VALUE='
              DELIMITED BY SIZE
              INTO SW-DATA-CONTENT
```

```
                WITH POINTER SW-TXT-PTR
        END-STRING.
        STRING CHAR-HANDLE
                DELIMITED BY SIZE
                INTO SW-DATA-CONTENT
                WITH POINTER SW-TXT-PTR
        END-STRING.
        STRING '>'
                DELIMITED BY SIZE
                INTO SW-DATA-CONTENT
                WITH POINTER SW-TXT-PTR
        END-STRING.
        STRING '<P>Enter your name and address in the fields below:'
                DELIMITED BY SIZE
                INTO SW-DATA-CONTENT
                WITH POINTER SW-TXT-PTR
        END-STRING.
        STRING '<P>Name:  <INPUT TYPE="text" NAME="name" SIZE=30'
                DELIMITED BY SIZE
                INTO SW-DATA-CONTENT
                WITH POINTER SW-TXT-PTR
        END-STRING.
        STRING ' VALUE="">'
                DELIMITED BY SIZE
                INTO SW-DATA-CONTENT
                WITH POINTER SW-TXT-PTR
        END-STRING.
        STRING '<P>Address:<INPUT TYPE="text" NAME="address"'
                DELIMITED BY SIZE
                INTO SW-DATA-CONTENT
                WITH POINTER SW-TXT-PTR
        END-STRING.
        STRING ' SIZE=70 VALUE="">'
                DELIMITED BY SIZE
                INTO SW-DATA-CONTENT
                WITH POINTER SW-TXT-PTR
        END-STRING.
        STRING '<P>Click here to register with the server:      <'
                DELIMITED BY SIZE
                INTO SW-DATA-CONTENT
                WITH POINTER SW-TXT-PTR
        END-STRING.
        STRING 'INPUT TYPE="submit" NAME="action" VALUE="REG   ">'
                DELIMITED BY SIZE
                INTO SW-DATA-CONTENT
                WITH POINTER SW-TXT-PTR
        STRING '<P>Click here to terminate the conversation:    '
                DELIMITED BY SIZE
                INTO SW-DATA-CONTENT
                WITH POINTER SW-TXT-PTR
        END-STRING.
        STRING '<INPUT TYPE="submit" NAME="action" VALUE="DELETE">'
                DELIMITED BY SIZE
                INTO SW-DATA-CONTENT
                WITH POINTER SW-TXT-PTR
        END-STRING.
        STRING '</FORM>'
                DELIMITED BY SIZE
                INTO SW-DATA-CONTENT
                WITH POINTER SW-TXT-PTR
        END-STRING.
    B010-IDENTIFY-BUILD-EXIT.
    C020-REGISTER-BUILD.
         MOVE LOW-VALUE TO SW-DATA-CONTENT.
         STRING '<HTML><HEAD>Shopping list</HEAD>'
                 DELIMITED BY SIZE
                 INTO SW-DATA-CONTENT
                 WITH POINTER SW-TXT-PTR
         END-STRING.
         STRING '<TITLE>Display Shopping List</TITLE</HEAD>'
                 DELIMITED BY SIZE
                 INTO SW-DATA-CONTENT
                 WITH POINTER SW-TXT-PTR
         END-STRING.
         STRING '<IMG SRC="http://ladoga.sanjose.ibm.com/slwhtml/'
                 DELIMITED BY SIZE
                 INTO SW-DATA-CONTENT
                 WITH POINTER SW-TXT-PTR
```

```
 END-STRING.
 STRING 'slwimg/cwmast.gif" ALT="CICSWEB Masthead" '
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
 END-STRING.
 STRING ' ALIGN="TOP">'
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
 END-STRING.
 STRING '<FORM METHOD=POST ACTION="/cgi-bin/REXXCIC3" >'
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
 END-STRING.
STRING '<INPUT TYPE="HIDDEN" NAME="function" '
         DELIMITED BY SIZE
         INTO SW-DATA-CONTENT
         WITH POINTER SW-TXT-PTR
STRING 'VALUE="ADD       ">'
         DELIMITED BY SIZE
         INTO SW-DATA-CONTENT
         WITH POINTER SW-TXT-PTR
END-STRING.
STRING '<INPUT TYPE="HIDDEN" NAME="name" VALUE='
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
END-STRING.
STRING ST-NAME
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
END-STRING.
STRING '><INPUT TYPE="HIDDEN" NAME="handle" VALUE='
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
END-STRING.
STRING CHAR-HANDLE
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
END-STRING.
STRING '>'
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
END-STRING.
STRING '><INPUT TYPE="HIDDEN"  NAME="address" VALUE='
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
END-STRING.
STRING ST-ADDRESS
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
END-STRING.
STRING '><P>There are no items currently on your '
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
END-STRING.
STRING 'shopping list. <P>'
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
END-STRING.
STRING 'Enter the item you wish to add to the list here:'
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
END-STRING.
STRING '<P>Item:  <INPUT TYPE="text" NAME="item" SIZE=30 '
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
```

```
           WITH POINTER SW-TXT-PTR
  END-STRING.
  STRING '<INPUT TYPE="submit" NAME="action" '
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
  END-STRING.
  STRING 'VALUE="ADD   "> <P>Click here to delete this'
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
  END-STRING.
  STRING 'list:  <INPUT TYPE="submit" NAME="action" '
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
  END-STRING.
  STRING 'VALUE="DELETE"> </FORM>'
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
  END-STRING.
 C020-REGISTER-BUILD-EXIT.
 D030-ADD-BUILD.
   MOVE LOW-VALUE TO SW-DATA-CONTENT
   STRING '<HTML><HEAD>Shopping list</HEAD>'
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
   END-STRING.
   STRING '<TITLE>Display Shopping List</TITLE</HEAD>'
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
   END-STRING.
   STRING '<IMG SRC="http://ladoga.sanjose.ibm.com/slwhtml/slw'
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
   END-STRING.
   STRING 'img/cwmast.gif" ALT="CICSWEB Masthead" ALIGN="TOP">'
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
   END-STRING.
   STRING '<FORM METHOD=POST ACTION="/cgi-bin/REXXCIC3" >'
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
   END-STRING.
  STRING '<INPUT TYPE="HIDDEN" NAME="function" '
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
  STRING 'VALUE="ADD       ">'
          DELIMITED BY SIZE
          INTO SW-DATA-CONTENT
          WITH POINTER SW-TXT-PTR
  END-STRING.
   STRING '<INPUT TYPE="HIDDEN" NAME="name" VALUE='
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
   END-STRING.
   STRING ST-NAME
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
```

```
           END-STRING.
           STRING '><INPUT TYPE="HIDDEN" NAME="handle" VALUE='
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
           END-STRING.
           STRING CHAR-HANDLE
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
           END-STRING.
           STRING '>'
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
             END-STRING.
           STRING '><INPUT TYPE="HIDDEN"  NAME="address" VALUE='
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
           END-STRING.
           STRING ST-ADDRESS
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
           END-STRING.
           IF ST-ITEMNUM NOT EQUAL 0 THEN
            PERFORM D030-GET-TS-QUEUE THRU D030-GET-TS-QUEUE-EXIT
           ELSE
            STRING '><P>There are no items currently on your shopping'
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
            END-STRING.
           STRING '<P>Enter the name of an item you wish to add'
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
           END-STRING.
           STRING ' to the list:  '
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
           END-STRING.
           STRING '<P>Item:  <INPUT TYPE="text" NAME="item" SIZE=30 VA'
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
           END-STRING.
           STRING 'LUE=""><P>Click here to add this input to the list:'
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
           END-STRING.
           STRING '<INPUT TYPE="submit" NAME="action" VALUE="ADD   ">'
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
           END-STRING.
           STRING '<P>Click here to delete this shopping list:  '
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
           END-STRING.
           STRING '<INPUT TYPE="submit" NAME="action" '                         '
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
           END-STRING.
           STRING 'VALUE="DELETE"></FORM>'
                    DELIMITED BY SIZE
                    INTO SW-DATA-CONTENT
                    WITH POINTER SW-TXT-PTR
           END-STRING.
       D030-ADD-BUILD-EXIT.
      *
       D030-GET-TS-QUEUE.
           STRING '><P>You currently have the following items on'
```

```
               DELIMITED BY SIZE
               INTO SW-DATA-CONTENT
               WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING ' your shopping list: <P>'
               DELIMITED BY SIZE
               INTO SW-DATA-CONTENT
               WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING '<UL>'
               DELIMITED BY SIZE
               INTO SW-DATA-CONTENT
               WITH POINTER SW-TXT-PTR
     END-STRING.
     PERFORM D030-READ-LOOP THRU D030-READ-LOOP-EXIT
     VARYING TEMPNUM
     FROM +1
     BY +1 UNTIL EIBRESP NOT EQUAL DFHRESP(NORMAL).
     STRING '</UL>'
               DELIMITED BY SIZE
               INTO SW-DATA-CONTENT
               WITH POINTER SW-TXT-PTR
     END-STRING.
     IF EIBRESP NOT EQUAL DFHRESP(ITEMERR) THEN
       STRING '<P>Error occurred reading TS queue.'
               DELIMITED BY SIZE
               INTO SW-DATA-CONTENT
               WITH POINTER SW-TXT-PTR
       END-STRING.
D030-GET-TS-QUEUE-EXIT.
D030-READ-LOOP.
               MOVE TSQ-LENGTH-VALUE  TO TSQ-LENGTH
               EXEC CICS READQ TS
                         QUEUE(ST-TSQ-NAME)
                         INTO(QUEUE-TEXT)
                         LENGTH(TSQ-LENGTH)
                         ITEM(TEMPNUM)
                         NOHANDLE
                         END-EXEC
     IF EIBRESP = DFHRESP(NORMAL) THEN
     STRING '<LI>'
               DELIMITED BY SIZE
               INTO SW-DATA-CONTENT
               WITH POINTER SW-TXT-PTR
     END-STRING
     STRING QUEUE-TEXT
               DELIMITED BY SPACES
               INTO SW-DATA-CONTENT
               WITH POINTER SW-TXT-PTR
     END-STRING
     END-IF.
D030-READ-LOOP-EXIT.
E040-DELETE-BUILD.
     MOVE LOW-VALUE TO SW-DATA-CONTENT.
     STRING '<HTML><HEAD><TITLE>NAME AND ADDRESS</TITLE></HEAD>'
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING '<IMG SRC="http://ladoga.sanjose.ibm.com/'
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
     STRING 'slwhtml/slwimg/cwmast.gif" ALT="CICSWEB Masthead">'
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
     STRING '<FORM METHOD=POST ACTION="/cgi-bin/REXXCIC3" >'
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING '<INPUT TYPE="HIDDEN" NAME="function" VALUE="">'
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
     END-STRING.
     STRING '<INPUT TYPE="HIDDEN" NAME="handle" VALUE="">'
```

```
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      END-STRING.
      STRING '<P>Your shopping list has been deleted'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      END-STRING.
      STRING '</FORM>'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      END-STRING.
E040-DELETE-BUILD-EXIT.
F050-INVALID-FUNCTION.
      STRING '<HTML><HEAD><TITLE>Invalid function</TITLE></HEAD>'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      END-STRING.
      STRING '<IMG SRC="http://ladoga.sanjose.ibm.com/'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      STRING 'slwhtml/slwimg/cwmast.gif" ALT="CICSWEB Masthead">'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      STRING '<P>SWLIST2 did not recognise the function with'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      END-STRING.
      STRING '<P> which it was called: *'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      END-STRING.
      STRING SW-FUNCTION
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      END-STRING.
      STRING '*'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      END-STRING.
      STRING '</FORM>'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      END-STRING.
      EXEC CICS RETURN END-EXEC.
F050-INVALID-FUNCTION-EXIT.
J070-ERROR.
      STRING '<HTML><HEAD><TITLE>Error Detected </TITLE></HEAD>'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      END-STRING.
      STRING '<IMG SRC="http://ladoga.sanjose.ibm.com/'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      STRING 'slwhtml/slwimg/cwmast.gif" ALT="CICSWEB Masthead">'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      STRING '<P>SWLIST2 detected the following error whilst'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
            WITH POINTER SW-TXT-PTR
      END-STRING.
      STRING ' processing this list: <P>'
            DELIMITED BY SIZE
            INTO SW-DATA-CONTENT
```

```
           WITH POINTER SW-TXT-PTR
      END-STRING.
      STRING MSG-TXT
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
      END-STRING.
      STRING '</FORM>'
           DELIMITED BY SIZE
           INTO SW-DATA-CONTENT
           WITH POINTER SW-TXT-PTR
      END-STRING.
      EXEC CICS RETURN END-EXEC.
 J070-ERROR-EXIT.
```

C.3 Assembler CICS/ESA State Management Program

```
         TITLE 'CICSSTAT - CICS/ESA SAMPLE STATE HANDLER       - ASSEMB*
               LER'
*********************************************************************
*                                                                   *
* MODULE NAME = CICSSTAT                                            *
*                                                                   *
* DESCRIPTIVE NAME = Module to handle CICS transactions that        *
*                    wish to have state maintained on their         *
*                    behalf.                                        *
*                    This program offers non terminal-oriented      *
*                    tasks, such as programs designed to interface  *
*                    with the Web, a facility to allow one CICS     *
*                    task to save information to be retrieved by    *
*                    another task using a unique identifier which   *
*                    is stored elsewhere, for example in a hidden   *
*                    field in a form.                               *
*                                                                   *
*                    The program can be run periodically to purge   *
*                    state data which has "timed out" if required,  *
*                    by running the CWBT transactions.              *
*                                                                   *
*                    The program can also be run to delete ALL      *
*                    state data resources                           *
*                                                                   *
*                    Allows callers to CREATE, RETRIEVE, UPDATE     *
*                    and DESTROY state information                  *
*                    When called as transaction CWBP, purges all    *
*                    state data which has not been updated for one  *
*                    hour.                                          *
*     COPYRIGHT = NONE                                              *
*                                                                   *
*********************************************************************
*
*        Commarea Structure
*
COMMAREA DSECT
COMM_START DS 0F
EYECATCH DS  CL4
FUNCTION DS  XL1
CREATE   EQU   X'C3'              * Create a new handle and state stg
RETRIEVE EQU   X'D9'               * Retrieve state info for a handle
STORE    EQU   X'E2'               * Store state info for a handle
DESTROY  EQU   X'C4'               * Destroy handle and state stg
RETCODE  DS    XL1
GOODRC   EQU   X'0'
BADCOMRC EQU   X'1'
INVFUNRC EQU   X'2'
NOSTGRC  EQU   X'3'
NOMATRC  EQU   X'4'
TSWERRC  EQU   X'5'
LENGERR  EQU   X'6'
NOSTMPRC EQU   X'7'
BADQNMRC EQU   X'8'
TIMEFRC  EQU   X'9'
TSWER2RC EQU   X'A'
RESERVED DS    XL2
HANDLE   DS    F                  * Unique conversation id
USERDATA DS    XL256              * User state information
COMLEN   EQU   *-COMM_START       * Length of COMMAREA
```

```
*
*
R7       EQU   7
WK1      EQU   6
WK2      EQU   8
COMPTR   EQU   5                  Pointer to Commarea
ANCHOR_PTR  EQU   10              Pointer to Anchor block
STATE_PTR  EQU   12                 Pointer to State Control Block
*
STATE_BLOCK        DSECT   State Control Block
STATE_EYECATCHER   DS   CL4
STATE_FORWARD_PTR  DS   F
STATE_BACKWARD_PTR DS   F
STATE_HANDLE       DS   F
STATE_TIMESTAMP    DS   PL8
STATE_USER_DATA    DS   XL256
STATE_LEN EQU *-STATE_BLOCK
*********************************************************************
*        This is the structure mapped on to our TS queue record       *
*        containing:                                                  *
*        1. Counter used to allocate unique IDs to state control      *
*           blocks.                                                   *
*        2. Anchors for the state control block                       *
*********************************************************************
ANCHOR_BLOCK   DSECT
ANCHOR_EYECATCHER DS  CL4
ANCHOR_COUNTER            DS F    Counter to generate unique handles
ANCHOR_FORWARD_PTR  DS F     Storage block chain forward ptr
ANCHOR_BACKWARD_PTR DS F     Storage block chain backward ptr
ANCHOR_TIMESTAMP DS PL8      Time at which this anchor was created
ANCHOR_LEN EQU *-ANCHOR_BLOCK  Length of anchor
         EJECT ,
DFHEISTG DSECT
*
*        Working Storage
*
TEMP_TIME DS   PL8
CURRENT_TIME DS   PL8
TEMP_STATE_PTR DS F
MESSAGES DS    CL80               TEMP STORE FOR MESSAGES
RESP     DS    1F                 RESPONSES TO CICS COMMANDS
TRANID   DS    CL4                TRANSACTION IDENTIFIER
TSLEN    DS    H
ITEMNUM  DS    H
FREE_SAVE_R7 DS F
GETMAIN_LENGTH DS F
ANCHOR_STG DS CL(ANCHOR_LEN)
         EJECT ,
CICSSTAT CSECT
         XC   MESSAGES,MESSAGES              Clear message field
*********************************************************************
*        Issue an ENQ so that we have the State Management lock       *
*********************************************************************
         EXEC CICS ENQ RESOURCE(STATE_MANAGER)                        *
                       LENGTH(L'STATE_MANAGER)                        *
                       RESP(RESP)
*********************************************************************
*   Call GETANCH to retrieve the TS record containing our anchor      *
*   information. If no address is returned, this must be the first    *
*   time we have been invoked, so call INIT_ANCHOR to get the         *
*   anchor block storage and initialise the TS record.                *
*********************************************************************
         LA    ANCHOR_PTR,ANCHOR_STG
         USING ANCHOR_BLOCK,ANCHOR_PTR
         USING STATE_BLOCK,STATE_PTR
         BAL   R7,GETANCH       Get the state control block anchor
         LTR   ANCHOR_PTR,ANCHOR_PTR  If the get for the anchor is OK
         BNZ   CHKFUNC          ..continue
         BAL   R7,INIT_ANCHOR   Else initialise the anchor
         LTR   ANCHOR_PTR,ANCHOR_PTR  If this fails as well........
         BZ    ENDFUNC          Give up !
CHKFUNC  DS    0H
         CLC   EIBTRNID,CWBT         Is it the timeout transaction ?
         BE    TIMEOUT_RTN           ..Yes, go do it.
         CLC   EIBTRNID,CWBP         Is it the purge transaction ?
         BE    PURGE_RTN             ..Yes, go do it.
*********************************************************************
*        If we get here, we have been linked to from another program. *
```

```
*        See whether there is a commarea for us to use.               *
*        If not, there is an error, as we should always be called     *
*        a commarea.                                                  *
*********************************************************************
         EXEC CICS ADDRESS COMMAREA(COMPTR)
         USING COMMAREA,COMPTR
         LTR   COMPTR,COMPTR
         BZ    BADCOM
*********************************************************************
*        Check the function code in the commarea to see why we have
*        been invoked, and call the appropriate routine.
*********************************************************************
         L     R7,=AL4(CREATE_RTN)  If function is create
         CLI   FUNCTION,CREATE      invoke the CREATE routine
         BE    CALLFUNC
*
         L     R7,=AL4(RETRIEVE_RTN) If function is retrieve
         CLI   FUNCTION,RETRIEVE     invoke the RETRIEVE routine
         BE    CALLFUNC
*
         L     R7,=AL4(STORE_RTN)    If function is store
         CLI   FUNCTION,STORE        invoke the STORE routine
         BE    CALLFUNC
*
         L     R7,=AL4(DESTROY_RTN)  If function is destroy
         CLI   FUNCTION,DESTROY      invoke the destroy routine
         BE    CALLFUNC
*********************************************************************
*        We should not get here. If we do, issue an error message     *
*********************************************************************
         B     INVFUNC
CALLFUNC DS    0H
         BR    R7            Invoke the requested routine
ENDFUNC  DS    0H
*********************************************************************
*       If an error was raised, issue message to operator             *
*********************************************************************
         CLI   RETCODE,GOODRC
         BE    ENDNOERR
         EXEC CICS WRITE OPERATOR TEXT(MESSAGES) TEXTLENGTH(L'MESSAGES)
ENDNOERR DS    0H
*********************************************************************
*        Issue an DEQ to release the State Management lock            *
*********************************************************************
         EXEC CICS DEQ RESOURCE(STATE_MANAGER)                        *
                       LENGTH(L'STATE_MANAGER)                        *
                       RESP(RESP)
         EXEC CICS RETURN
         EJECT ,
CREATE_RTN DS 0H
*********************************************************************
*        This routine does the following:                             *
*        1. Creates a new handle for a conversation                   *
*        2. Acquires storage for a new state block                    *
*                                                                     *
*        Increment the unique handle generator count by 1             *
*********************************************************************
         L     WK1,ANCHOR_COUNTER   Get the current count
         LA    WK1,1(,WK1)         Increment it by one
         ST    WK1,ANCHOR_COUNTER   Put it back
*
*        Get storage for the new state block
*
         MVC   GETMAIN_LENGTH,=AL4(STATE_LEN)
         EXEC CICS GETMAIN SHARED SET(STATE_PTR) RESP(RESP)           *
               FLENGTH(GETMAIN_LENGTH) INITIMG(X'00')
         CLC   RESP,DFHRESP(NORMAL)    Storage acquired OK ?
         BNE   NOSTG                   No, check for no queue
*********************************************************************
*        Add new state block to the chain                            *
*********************************************************************
         L    WK1,ANCHOR_BACKWARD_PTR
         LTR  WK1,WK1
         BNZ  ADDELEM
*********************************************************************
*        No elements currently in chain                              *
*********************************************************************
         ST    STATE_PTR,ANCHOR_BACKWARD_PTR
```

```
         ST    STATE_PTR,ANCHOR_FORWARD_PTR
         B     CONTINUE
*********************************************************************
*        Add element to existing chain                              *
*********************************************************************
ADDELEM  DS   0H
         L    WK1,ANCHOR_BACKWARD_PTR
         ST   STATE_PTR,STATE_FORWARD_PTR-STATE_BLOCK(,WK1)
         XC   STATE_FORWARD_PTR,STATE_FORWARD_PTR
         ST   WK1,STATE_BACKWARD_PTR
         ST   STATE_PTR,ANCHOR_BACKWARD_PTR
CONTINUE DS    0H
*********************************************************************
*        Write the anchor block back to TS                          *
*********************************************************************
         BAL   R7,REWRITE
*********************************************************************
*        Initialise element                                         *
*********************************************************************
         MVC  STATE_EYECATCHER,STATE_EYE_INIT
         BAL  R7,TIMESTAMP
         MVC  STATE_HANDLE,ANCHOR_COUNTER
         MVC  STATE_USER_DATA,USERDATA
*********************************************************************
*        Put results into commarea to pass back to caller           *
*********************************************************************
         MVC  HANDLE,STATE_HANDLE
         MVI  RETCODE,GOODRC
         B     ENDFUNC
*
*
RETRIEVE_RTN DS 0H
*********************************************************************
*        This routine does the following:                           *
*        1. Retrieve the state data for a handle passed in the      *
*           Commarea.                                               *
*        2. Update the timestamp in the state block.                *
*        3. Return the state data to the application                *
*********************************************************************
         BAL  R7,LOCATE_STATE_BLOCK
         LTR  STATE_PTR,STATE_PTR
         BZ   RETRIEVE_NO_MATCH
         BAL  R7,TIMESTAMP
         MVC  USERDATA,STATE_USER_DATA
         B   ENDFUNC
*********************************************************************
*      No match was found for the handle passed by the caller.      *
*      Either it has timed out, or there is a logic error.          *
*********************************************************************
RETRIEVE_NO_MATCH DS 0H
         B   NOMATCH
*
*
*
STORE_RTN DS 0H
*********************************************************************
*        This routine does the following:                           *
*        1. Finds the requested handle                              *
*        2. Updates the timestamp                                   *
*        3. Stores the new state data in the block                  *
*********************************************************************
         BAL  R7,LOCATE_STATE_BLOCK
         LTR  STATE_PTR,STATE_PTR
         BZ   STORE_NO_MATCH
         BAL  R7,TIMESTAMP
         MVC  STATE_USER_DATA,USERDATA
         MVI  RETCODE,GOODRC
         B ENDFUNC
STORE_NO_MATCH DS 0H
         B NOMATCH
*
DESTROY_RTN DS 0H
*********************************************************************
*        This routine does the following:                           *
*        1. Finds the requested block                               *
*        2. Calls FREE_STATE_BLOCK to destroy the element.          *
*        3. Freemains the state block                               *
*********************************************************************
```

```
         BAL  R7,LOCATE_STATE_BLOCK
         LTR  STATE_PTR,STATE_PTR
         BZ   DESTROY_NO_MATCH
         BAL  R7,FREE_STATE_BLOCK
         MVI  RETCODE,GOODRC
         B ENDFUNC
DESTROY_NO_MATCH DS 0H
         B NOMATCH
TIMEOUT_RTN  DS    0H
*********************************************************************
*        Routine which loops through all the state control blocks     *
*        and deletes those which have been around for more than       *
*        1 hour (change EXPIRY_INTERVAL if you wish to                *
*        customize this timeout value).                               *
*********************************************************************
         EXEC CICS ASKTIME ABSTIME(CURRENT_TIME) RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)    Timestamp made OK ?
         BNE   TIMEFAIL
         L     STATE_PTR,ANCHOR_FORWARD_PTR     Initialise state ptr
TIMELOOP DS    0H
         LTR   STATE_PTR,STATE_PTR              If nulls, we are done
         BZ    ALLDONE
         MVC   TEMP_TIME,STATE_TIMESTAMP      If (last update time +
         AP    TEMP_TIME,EXPIRY_INTERVAL            expiry interval)
         CP    TEMP_TIME,CURRENT_TIME             is less than current
         BNL   TIMEOK                          then remove it
         MVC   TEMP_STATE_PTR,STATE_FORWARD_PTR else remove this block
         BAL   R7,FREE_STATE_BLOCK
         L     STATE_PTR,TEMP_STATE_PTR
         B     TIMELOOP
TIMEOK   DS    0H
         L     STATE_PTR,STATE_FORWARD_PTR
ALLDONE  DS     0H
         B     ENDFUNC
PURGE_RTN  DS    0H
*********************************************************************
*        Routine which loops through all the state control blocks     *
*        and deletes them, then deletes the anchor block.             *
*********************************************************************
PURGE_LOOP DS    0H
         L     STATE_PTR,ANCHOR_FORWARD_PTR     Initialise state ptr
         LTR   STATE_PTR,STATE_PTR
         BZ    PURGE_DONE
         BAL   R7,FREE_STATE_BLOCK
         B     PURGE_LOOP
PURGE_DONE DS     0H
         EXEC CICS DELETEQ TS QUEUE(WEBQUEUE) RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)    Timestamp made OK ?
         BNE   DELEFAIL
         EXEC CICS WRITE OPERATOR TEXT(TERMMSG) TEXTLENGTH(L'TERMMSG)
         B     ENDFUNC
GETANCH  DS    0H
*********************************************************************
*        Retrieve the TS Q record containing our anchor block         *
*        if it's not there, create a new one                          *
*********************************************************************
         SR    WK1,WK1
         LA    WK1,1(,WK1)
         STCM  WK1,B'0011',ITEMNUM
         MVC   TSLEN,=AL2(ANCHOR_LEN)
         EXEC CICS READQ TS QUEUE(WEBQUEUE) INTO(ANCHOR_BLOCK)        *
         LENGTH(TSLEN) ITEM(ITEMNUM) RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)    Record read OK ?
         BNE   NOQUEUE                 No, check for no queue
         LA    ANCHOR_PTR,ANCHOR_BLOCK
         BR    R7
NOQUEUE  DS    0H
         SR    ANCHOR_PTR,ANCHOR_PTR
         CLC   RESP,DFHRESP(QIDERR)    No queue found ?
         BNE   BADQNM                  No, unexpected error
         BR    R7                      Return to the caller
INIT_ANCHOR DS  0H
         LA    ANCHOR_PTR,ANCHOR_STG
*********************************************************************
*        This routine is responsible for acquiring and initialising   *
*        the state control block anchor. It is a TS record,           *
*        written the first time we are invoked.                       *
*********************************************************************
```

```
         XC    ANCHOR_BLOCK,ANCHOR_BLOCK
         MVC   ANCHOR_EYECATCHER,ANCHOR_EYE_INIT
         MVC   TSLEN,=AL2(ANCHOR_LEN)
         EXEC CICS ASKTIME ABSTIME(ANCHOR_TIMESTAMP) RESP(RESP)
         EXEC CICS WRITEQ TS QUEUE(WEBQUEUE) RESP(RESP)                *
               FROM(ANCHOR_BLOCK)                                      *
               LENGTH(TSLEN)
         CLC   RESP,DFHRESP(NORMAL)    Record written OK ?
         BNE   WRITERR                 No, check for no queue
         EXEC CICS WRITE OPERATOR TEXT(INITMSG) TEXTLENGTH(L'INITMSG)
         BR    R7
WRITERR  DS    0H
         SR    ANCHOR_PTR,ANCHOR_PTR
         B     TSWERR
         BR    R7                      Return to the caller
LOCATE_STATE_BLOCK DS 0H
*********************************************************************
* Subroutine to locate state block using handle from commarea       *
* If we find it, load the address into STATE_PTR. Otherwise set     *
* STATE_PTR to nulls before returning.                              *
*********************************************************************
         L   STATE_PTR,ANCHOR_FORWARD_PTR
         LTR STATE_PTR,STATE_PTR         See if list is empty
         BZ  LOCATE_NO_MATCH
LOCATE_SEARCH_LOOP DS 0H
         L   WK1,HANDLE
         C   WK1,STATE_HANDLE
         BE  LOCATE_FOUND
         L   STATE_PTR,STATE_FORWARD_PTR
         LTR STATE_PTR,STATE_PTR
         BZ  LOCATE_NO_MATCH
         B   LOCATE_SEARCH_LOOP
LOCATE_FOUND DS 0H
         BR  R7
LOCATE_NO_MATCH DS 0H
         SR  STATE_PTR,STATE_PTR
         BR  R7
FREE_STATE_BLOCK DS 0H
*********************************************************************
* Subroutine to free a state block, unchaining it from the list,    *
* and then FREEMAIN the storage.                                    *
*********************************************************************
         L     WK1,STATE_FORWARD_PTR
         LTR   WK1,WK1
         BZ    FREE1
         MVC   STATE_BACKWARD_PTR-STATE_BLOCK(4,WK1),STATE_BACKWARD_PTR
         B     FREE2
FREE1    DS    0H
         L     WK1,STATE_BACKWARD_PTR
         ST    WK1,ANCHOR_BACKWARD_PTR
FREE2    DS    0H
*********************************************************************
*        Sort out forward pointers                                  *
*********************************************************************
         L     WK1,STATE_BACKWARD_PTR
         LTR   WK1,WK1
         BZ    FREE3
         MVC   STATE_FORWARD_PTR-STATE_BLOCK(,WK1),STATE_FORWARD_PTR
         B     FREE4
FREE3    DS    0H
         L     WK1,STATE_FORWARD_PTR
         ST    WK1,ANCHOR_FORWARD_PTR
FREE4    DS    0H
*********************************************************************
*        Freemain the state block storage                           *
*********************************************************************
         EXEC CICS FREEMAIN DATAPOINTER(STATE_PTR) RESP(RESP)
         ST    R7,FREE_SAVE_R7              Save R7 across call
         BAL   R7,REWRITE                   to update anchor
         L     R7,FREE_SAVE_R7              Restore R7
         BR    R7                           Return
*********************************************************************
*        Subroutine to timestamp a state block                      *
*********************************************************************
TIMESTAMP DS 0H
         EXEC CICS ASKTIME ABSTIME(STATE_TIMESTAMP) RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)    Timestamp made OK ?
         BNE   NOSTMP                  No, raise error.
```

```
         BR R7
REWRITE  DS    0H
*****************************************************************
*        Subroutine to write updated anchor block back to TS        *
*****************************************************************
         MVC   ITEMNUM,=AL2(1)
         MVC   TSLEN,=AL2(ANCHOR_LEN)
         EXEC CICS WRITEQ TS QUEUE(WEBQUEUE) RESP(RESP)               *
               FROM(ANCHOR_BLOCK)                                     *
               LENGTH(TSLEN)                                          *
               ITEM(ITEMNUM)                                          *
               REWRITE
         CLC   RESP,DFHRESP(NORMAL)    Record written OK ?
         BNE   TSWERR                  No, raise error
         BR    R7
         EJECT ,
*****************************************************************
*        Error Routines                                           *
*****************************************************************
BADCOM   DS    0H                  DUPLICATE RECORD
         MVC   MESSAGES,=CL(L'MESSAGES)'Invalid Commarea passed.'
         MVI   RETCODE,BADCOMRC
         B     ENDFUNC               COMPLETE, GO FINISH
INVFUNC  DS    0H                  DUPLICATE RECORD
         MVC   MESSAGES,=CL(L'MESSAGES)'Invalid function passed.'
         MVI   RETCODE,INVFUNRC
         B     ENDFUNC               COMPLETE, GO FINISH
BADQNM   DS    0H
         MVC   MESSAGES,=CL(L'MESSAGES)'Unable to retrieve state block *
               anchor.'
         MVI   RETCODE,BADQNMRC
         B     ENDFUNC
TSWERR   DS    0H
         MVC   MESSAGES,=CL(L'MESSAGES)'Initial TS write for state bloc*
               k anchor failed.'
         MVI   RETCODE,TSWERRC
         B     ENDFUNC
TSWERR2  DS    0H
         MVC   MESSAGES,=CL(L'MESSAGES)'Update of anchor failed.      '
         MVI   RETCODE,TSWER2RC
         B     ENDFUNC
NOSTG    DS    0H
         MVC   MESSAGES,=CL(L'MESSAGES)'Getmain for state block failed.*
                      '
         B     ENDFUNC
NOSTMP   DS    0H
         MVC   MESSAGES,=CL(L'MESSAGES)'Timestamp for state block faile*
               d'
         MVI   RETCODE,NOSTMPRC
         B     ENDFUNC
NOMATCH  DS    0H
         MVC   MESSAGES,=CL(L'MESSAGES)'State data could not be found f*
               or the handle supplied'
         MVI   RETCODE,NOMATRC
         B     ENDFUNC
LENGERR2 DS    0H
         MVC   MESSAGES,=CL(L'MESSAGES)'Read from TSQ returned bad leng*
               th'
         SR    ANCHOR_PTR,ANCHOR_PTR
         MVI   RETCODE,LENGERR
         B     ENDFUNC
TIMEFAIL DS    0H
         MVC   MESSAGES,=CL(L'MESSAGES)'Unable to get current time.    *
                 '
         B     ENDFUNC
DELEFAIL DS    0H
         MVC   MESSAGES,=CL(L'MESSAGES)'Unable to delete State Block An*
               chor.   '
         B     ENDFUNC
*
*
*        CONSTANTS
*
EXPIRY_INTERVAL DC XL8'00000000360000C'  * 1 hour in absolute time
ANCHOR_EYE_INIT   DC CL4'SCBA'
STATE_EYE_INIT DC CL4'SCBE'
CWBP     DC    CL4'CWBP'
CWBT     DC    CL4'CWBT'
```

```
ANCHORID DC    CL8'CICSSTAT'
WEBQUEUE DC    CL8'CICSWEB '
STATE_MANAGER DC  CL13'STATE_MANAGER'
INITMSG DC     CL80'State data manager CICSSTAT initialized'
TERMMSG DC     CL80'State data manager CICSSTAT terminated'
*
         LTORG
         END
```

Appendix D. CICS/ESA Sockets Application Sample

This chapter contains the source code for the sample CICS/ESA Sockets sample.

- C CGI script
- COBOL CICS/ESA Web server application program
- MVS JCL to compile a COBOL program

D.1 C CGI Script

This script is in two parts:

client.c is the main CGI script.

sockets.c contains sockets functions used by client.c.

D.1.1 client.c

```
/*
 * PROGRAM:     client1.c
 *
 * DESCRIPTION: This is the client part of the CICS sockets web sample.
 *              client1.c contains all the CGI code for the client.  It
 *              is linked with sockets.c to provide a CGI program which
 *              can issues TCP/IP socket calls.
 *
 *              This program sends an initial FORM to the web browser,
 *              receives input from the web-browser, sends this information
 *              to the server using socket calls, and then sends the info
 *              from the server back to the web browser and waits for input
 *              again.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define LF      10
#define HTML_BREAK      printf("<P>%c", LF);
typedef struct {
    char name[128];
    char val[128];
} entry;

typedef struct {
    char qfield[256];
    int  qlen;
    char qname[256];
} fields;

static fields idxfields[] = { {"XTran", 4, "Tranid"},
                              {"XAddr", 20, "Address"},
                              {"XPort", 5, "Port"},
                            {"", 0, ""}
                          };

/* function prototypes */
void send_doc(int which);
void getword(char *word, char *line, char stop);
void plustospace(char *str);
char *do_socket_call(char *addr, char *port, char *tran, char *action,
                     char *queue, char *qdata);

/*
 * getword()
 *
 * This function parses the HTML text received from the form, and returns
 * the next word (ie. up to the next 'stop' character)
```

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

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

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

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

/*
 * plustospace()
 *
 * This function converts all '+' characters to spaces in the input text
 * from the form.
 *
 */
void plustospace(char *str)
{
    register int x;

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

/*
 * send_doc()
 *
 * Sends the initial HTML form
 *
 */
void send_doc(int which) {
    int x;

    printf("<TITLE>CICS Sockets CGI</TITLE>%c", LF);
    printf("<H1>CICS Sockets CGI</H1>%c", LF);
    HTML_BREAK
    printf("<HR>%c", LF);

    printf("Enter the tranid, host IP address and port to connect to:%c", LF);
    printf("<FORM ACTION=\"http://%s:%s%s\">%c", getenv("SERVER_NAME"),
           getenv("SERVER_PORT"), getenv("SCRIPT_NAME"), LF);

    HTML_BREAK

    printf("%s ", idxfields[0].qname);
    printf("<INPUT TYPE=\"text\" NAME=\"%s\" MAXLENGTH=\"%d\" SIZE=\"%d\">%c"
               ,idxfields[0].qfield, idxfields[0].qlen
               ,idxfields[0].qlen+1,LF);

    printf("    %s ", idxfields[1].qname);
    printf("<INPUT TYPE=\"text\" NAME=\"%s\" MAXLENGTH=\"%d\" SIZE=\"%d\">%c"
               ,idxfields[1].qfield, idxfields[1].qlen
               ,idxfields[1].qlen,LF);

    printf("    %s ", idxfields[2].qname);
    printf("<INPUT TYPE=\"text\" NAME=\"%s\" MAXLENGTH=\"%d\" SIZE=\"%d\">%c"
               ,idxfields[2].qfield, idxfields[2].qlen
               ,idxfields[2].qlen,LF);

    printf("<INPUT TYPE=\"hidden\" NAME=\"XQueue\" VALUE=\"\">%c",LF);

    HTML_BREAK
    printf("<HR>%c", LF);
    HTML_BREAK

    printf("<INPUT TYPE=\"submit\" VALUE=\"Click here to send request\">%c", LF);
    printf("  <INPUT TYPE=\"reset\"  VALUE=\"Clear entry\">%c", LF);
    printf("</FORM>%c", LF);

}
```

```
main(int argc, char *argv[]) {
    char message[2001];
    char CICSTran[5];
    char CICSAddr[21];
    char CICSPort[6];
    char CICSAction[7];
    char CICSQueue[9];
    char CICSQData[51];
    entry entries[256];
    register int x,m=0;
    char *cl;
    char *tp;
    char returnstr[1024], typestr[4098], commandstr[8192], serverstr[256];

    printf("Content-type: text/html%c%c",LF,LF);

    cl = getenv("QUERY_STRING");

    if((!cl) || (!cl[0])) {
        send_doc(0);
        exit(1);
    }

    if (cl[0] != '\0')
    {
       getword(returnstr, cl, '&');
       getword(typestr, returnstr, '=');
       strcpy(CICSTran, returnstr);

       getword(returnstr, cl, '&');
       getword(typestr, returnstr, '=');
       strcpy(CICSAddr, returnstr);

       getword(returnstr, cl, '&');
       getword(typestr, returnstr, '=');
       strcpy(CICSPort, returnstr);

       getword(returnstr, cl, '&');
       getword(typestr, returnstr, '=');
       strcpy(CICSAction, returnstr);

       getword(returnstr, cl, '&');
       getword(typestr, returnstr, '=');
       strcpy(CICSQueue, returnstr);

       getword(returnstr, cl, '&');
       getword(typestr, returnstr, '=');
       plustospace(returnstr);
       strcpy(CICSQData, returnstr);

    }

    printf("<HTML>%c<BODY>%c", LF,LF);
    printf("<H1>Query Results</H1>%c", LF);
    HTML_BREAK

    CICSTran[4]='\0';    /* ensure null-termination of strings */
    CICSAddr[20]='\0';
    CICSPort[5]='\0';
    CICSAction[1]='\0';
    CICSQueue[8]='\0';
    CICSQData[50]='\0';

    /*
     * Here we call do_socket_call() passing the user information.  This
     * passes the info the server, and returns the HTML information which
     * the server sends back.
     * This return info is then displayed.
     * The server has control over what is sent back and hence what is
     * displayed.
     */
    strcpy(message, do_socket_call((char *)CICSAddr, (char *)CICSPort,
                                   (char *)CICSTran, (char *)CICSAction,
                                   (char *)CICSQueue, (char *)CICSQData));

    printf("%s  %c", message, LF);
```

```
    printf("</BODY>%c</HTML>", LF);
}
```

D.1.2 sockets.c

```
/*
 * PROGRAM:     sockets.c
 *
 * DESCRIPTION: This code provides the sockets functions which are called
 *              to converse with the CICS server transaction
 *
 */

                          /* HEADER FILES                              */
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>   /* Socket details                            */
#include <stdio.h>
#include <sys/errno.h>       /* To enable network error trapping        */

#include <string.h>

                          /* CONSTANTS                                 */
#define io_buf_size  2001 /* data buffer for socket comms, size      */
#define recv_flags      0 /* flags parameter on recv()               */
#define send_flags      0 /* flags parameter on send()               */
#define socket_protocol 0 /* protocol parameter on socket()          */
#define call_failed    -1 /* most TCP calls return -1 if failed      */

struct hostent *gethostbyname(char *name);
int    clients_sock_desc;       /* Client's socket                   */
struct hostent *hostnm_info; /* Server hostname information         */
struct sockaddr_in server_inetaddr; /* Server internet address    */
unsigned short clients_port; /* Port number client will bind to   */

char   transid[5];
char   tsqueue[9];
char   tsact[2];
char   tsqdata[51];
char   padding[50] = "                                                  ";

int    socket(int domain, int type, int protocol);
int    connect(int s, struct sockaddr *name, int namelen);
int    close(int s);
void   report_nerrno(int error_number);
char  *do_read();
void   shut_down();

void exit(int status);

int atoi(char *string);

/*
 * do_socket_call()
 *
 * Function called from main() to send data to server, and await reply
 */
char *do_socket_call(char *addr, char *port, char *tran, char *action,
                     char *queue, char *qdata)
{

   char input[10];
   int recv_len;

   clients_port = (unsigned short) atoi(port);
   server_inetaddr.sin_family      = AF_INET;
   server_inetaddr.sin_port        = htons(clients_port);
   server_inetaddr.sin_addr.s_addr = (inet_addr(addr));

   strcpy(transid,tran);
   transid[4] = '\0';

   strcpy(tsact,action);
   tsact[1] = '\0';

   strcpy(tsqueue,queue);
   tsqueue[8] = '\0';
```

```
   strcpy(tsqdata,qdata);
   tsqdata[50] = '\0';

   return (char *)do_read();
}

/*
 * startup()
 *
 * Initialise the connection with the CICS server, and send the initial
 * data from the client.
 * The limit is 30 bytes on the first send.  If the user requested a WRITE
 * operation, then there is more data, so we must perform further socket
 * calls to transfer the info.
 */
void startup(char *start_buf)
{
   char io_buf[io_buf_size];
   int    num_bytes;
   int    io_len;

   unsigned char *x;
   int y;

   /* Proper error conditions need to be handled here */

   if ((clients_sock_desc =
        socket(AF_INET,SOCK_STREAM,socket_protocol)) == call_failed)
   {
     printf("Problem with socket()\n");
     report_nerrno(errno);
     exit(3);
   }

   if (connect(clients_sock_desc, (struct sockaddr *) &server_inetaddr,
       sizeof(server_inetaddr)) == call_failed)
   {
     printf("Problem with connect()\n");
     printf("errno = %d\n",errno);
     report_nerrno(errno);
     exit(4);
   }

   /* printf("\n-------> Connection opened.\n\n"); */

   strcpy(io_buf,start_buf);
   io_len = strlen(io_buf);

   if (send(clients_sock_desc,io_buf,io_len,send_flags) == call_failed)
   {
     printf("Problem with send()\n");
     report_nerrno(errno);
     exit(5);
   }

   return;

}  /* end of startup() */


/*
 * do_read()
 *
 * Get the HTML information back from the CICS server transaction
 */
char *do_read()
{
   /* function to read msg msg_num for user to_user             */
   char from_user[9];
   char to_user[9];
   char subject[21];
   char message[433];
   char rcv_buf[io_buf_size+1];
   char io_buf[io_buf_size+1];
   int    num_bytes;
   int    io_len;
   char *next_field;
```

```
   strcpy(io_buf,transid);
   strcat(io_buf,",");
   strncat(tsact,padding,(1-strlen(tsact)));
   strcat(io_buf,tsact);
   strncat(tsqueue,padding,(8-strlen(tsqueue)));
   strcat(io_buf,tsqueue);

   startup(io_buf);

   if (!strcmp(tsact,"W"))
   {
      /* send MORE bytes to the server - the data to write to the queue */
      /* first, receive a response from the server (help keep things in */
      /* sync, and also flushes the buffers correctly :-)               */

      num_bytes = recv(clients_sock_desc,rcv_buf,io_buf_size,recv_flags);
      if (num_bytes == call_failed)
      {
        printf("Problem with recv()\n");
        report_nerrno(errno);
        exit(6);
      }

      strncat(tsqdata,padding,(50-strlen(tsqdata)));
      strcpy(io_buf,tsqdata);
      io_len = strlen(io_buf);
      if (send(clients_sock_desc,io_buf,io_len,send_flags) == call_failed)
      {
        printf("Problem with send()\n");
        report_nerrno(errno);
        exit(5);
      }

      num_bytes = recv(clients_sock_desc,rcv_buf,io_buf_size,recv_flags);
      if (num_bytes == call_failed)
      {
        printf("Problem with recv()\n");
        report_nerrno(errno);
        exit(6);
      }

   }

   num_bytes = recv(clients_sock_desc,rcv_buf,io_buf_size,recv_flags);
   if (num_bytes == call_failed)
   {
     printf("Problem with recv()\n");
     report_nerrno(errno);
     exit(6);
   }

   shut_down();
   return(rcv_buf);
}

/*
 * shut_down()
 *
 * closes the connection with the CICS server
 */
void shut_down()
{
   if ((close(clients_sock_desc)) == call_failed)
   {
     printf("Problem with close()\n");
     report_nerrno(errno);
     exit(8);
   }
   return;
}


/* Reports network errors                                            */
void report_nerrno(int error_number)
{
  switch (error_number)
  {
```

```
      case ENOTSOCK :
        printf("ENOTSOCK - invalid socket descriptor\n");
        break;
      case EOPNOTSUPP :
        printf("EOPNOTSUPP - socket descriptor does not support listen()\n");
        break;
      case EADDRINUSE :
        printf("EADDRINUSE - address already in use\n");
        break;
      case EADDRNOTAVAIL :
        printf("EADDRNOTAVAIL - address specified invalid on this host\n");
        break;
      case EAFNOSUPPORT :
        printf("EAFNOSUPPORT - address family is not supported\n");
        break;
      case EINVAL :
        printf("EINVAL - socket already bound/unexpected namelen length\n");
        break;
      case ENOBUFS :
        printf("ENOBUFS - no buffer space available\n");
        break;
      case EPROTONOSUPPORT :
        printf("EPROTONOSUPPORT - protocol unsupported by this\n");
        printf("                  domain/socket type\n");
        break;
      case EPROTOTYPE :
        printf("EPROTOTYPE - wrong type of protocol for this socket\n");
        break;
      case EWOULDBLOCK :
        printf("EWOULDBLOCK - the socket is in nonblocking mode\n");
        printf("              & no data is available to read\n");
        break;
      case EALREADY :
        printf("EALREADY - the socket is nonblocking & a previous\n");
        printf("           connection attempt is incomplete\n");
        break;
      case ENOTCONN :
        printf("ENOTCONN - the socket is not connected\n");
        break;
      case ECONNREFUSED :
        printf("ECONNREFUSED - the connection request was rejected by\n");
        printf("               the destination host\n");
        break;
      case EINPROGRESS :
        printf("EINPROGRESS - the socket is marked non-blocking & the\n");
        printf("              connection cannot be made immediately\n");
        printf("            - this is not an error condition\n");
        break;
      case EISCONN :
        printf("EISCONN - the socket is already connected\n");
        break;
      case ENETUNREACH :
        printf("ENETUNREACH - network cannot be reached from this host\n");
        break;
      case ETIMEDOUT :
        printf("ETIMEDOUT - the connection establishment timed out\n");
        printf("            before a connection was made\n");
    }
  }
```

D.2 COBOL CICS/ESA Web Server Application Program

```
       * $SEG(TCPSERV1)
       *---------------------------------------------------------------*
       *                                                               *
       *   MODULE NAME : TCPSERV1                                      *
       *                                                               *
       *   DESCRIPTION :                                               *
       *                                                               *
       *      THIS IS THE SERVER PART OF THE CICS SOCKETS WEB SAMPLE.  *
       *      THIS TRANSACTION IS STARTED BY THE CICS SOCKETS          *
       *      LISTENER TRANSACTION WHEN A REQUEST IS RECEIVED FROM     *
       *      THE WEB CLIENT PROGRAM.  IT DECODES THE INFORMATION ON   *
       *      THE REQUEST AND PERFORMS THE REQUIRED ACTION ON THE      *
       *      SPECIFIED CICS TS QUEUE.                                 *
       *                                                               *
```

```
*      WHEN THE ACTION HAS COMPLETED, THIS TRANSACTION SENDS   *
*      BACK HTML DATA TO THE CLIENT, WHICH CONTAINS THE        *
*      RESULT OF THE ACTION.  THE CLIENT CAN THEN DISPLAY      *
*      THIS DATA ON THE WEB BROWSER.                           *
*                                                              *
*      THE SOCKET CALLS ISSUED BY THIS PROGRAM ARE:            *
*                                                              *
*         . TAKESOCKET - ACQUIRES THE SOCKET PASSED BY THE     *
*                        LISTENER TRANSACTION                  *
*                                                              *
*         . RECEIVE   - GET MORE DATA FROM THE CLIENT          *
*                                                              *
*         . WRITE      - SEND HTML DATA BACK TO CLIENT         *
*                                                              *
*                                                              *
*--------------------------------------------------------------*
*
 IDENTIFICATION DIVISION.
 PROGRAM-ID. TCPSERV1.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
*
 WORKING-STORAGE SECTION.
 77  FIRST-REQUEST                 PIC X(24)
      VALUE IS ' FIRST REQUEST ......  '.
 77  SECOND-REQUEST               PIC X(24)
      VALUE IS ' SECOND REQUEST .....  '.
 77  TAKE-ERR                      PIC X(24)
      VALUE IS ' TAKESOCKET FAIL       '.
 77  TAKE-SUCCESS                   PIC X(24)
      VALUE IS ' TAKESOCKET SUCCESSFUL '.
 77  SOCK-SAME                     PIC X(24)
      VALUE IS ' TAKESOCKET SOCKET SAME'.
 77  SOCK-DIFF                      PIC X(24)
      VALUE IS ' TAKESOCKET SOCKET DIFF'.
 77  READ-ERR                      PIC X(24)
      VALUE IS ' READ SOCKET FAIL      '.
 77  READ-SUCCESS                  PIC X(24)
      VALUE IS ' READ SOCKET SUCCESSFUL '.
 77  WRITE-ERR                     PIC X(24)
      VALUE IS ' WRITE SOCKET FAIL     '.
 77  WRITE-END-ERR                      PIC X(32)
      VALUE IS ' WRITE SOCKET FAIL - PGM END MSG'.
 77  WRITE-SUCCESS                 PIC X(25)
      VALUE IS ' WRITE SOCKET SUCCESSFUL '.
 77  CLOS-ERR                      PIC X(24)
      VALUE IS ' CLOSE SOCKET FAIL     '.
 77  CLOS-SUCCESS                 PIC X(24)
      VALUE IS 'CLOSE SOCKET SUCCESSFUL '.
 77  INVREQ-ERR                    PIC X(24)
      VALUE IS 'INTERFACE IS NOT ACTIVE '.
 77  IOERR-ERR                    PIC X(24)
      VALUE IS 'IOERR OCCURRS          '.
 77  IOCTL-ERR                    PIC X(24)
      VALUE IS 'IOCTL ERROR            '.
 77  LENGERR-ERR                  PIC X(24)
      VALUE IS 'LENGERR ERROR          '.
 77  ITEMERR-ERR                  PIC X(24)
      VALUE IS 'ITEMERR ERROR          '.
 77  NOSPACE-ERR                  PIC X(24)
      VALUE IS 'NOSPACE CONDITION      '.
 77  QIDERR-ERR                 PIC X(24)
      VALUE IS 'QIDERR  CONDITION      '.
 77  ENDDATA-ERR                PIC X(30)
      VALUE IS 'RETRIEVE DATA CAN NOT BE FOUND'.
 77  WRKEND                     PIC X(20)
      VALUE 'CONNECTION END     '.

 01  WRKMSG.
     02 WRKM                        PIC X(15)
        VALUE IS 'DATA WRITTEN OK'.

 01  MSG-IN.
     02 QUEUE-ACTION                PIC X.
     02 QUEUE-ID                    PIC X(8).

 01  NEW-TASK-START.
     02 TITLE1                      PIC X(32)
```

```
VALUE IS '<TITLE>CICS Sockets CGI</TITLE>'.
02 FORM1.
   03 FORM1-PART1              PIC X(6)
   VALUE IS '<FORM '.
   03 FORM1-PART2              PIC X(37)
   VALUE IS 'ACTION="http://fussy.hursley.ibm.com:'.
   03 FORM1-PART3              PIC X(21)
   VALUE IS '80/cgi-bin/cicscgi">'.

02 HIDDEN1.
   03 TYPE1                    PIC X(21)
      VALUE IS '<INPUT TYPE="hidden" '.
   03 NAME1                    PIC X(13)
      VALUE IS 'NAME="XTran" '.
   03 VALUE1                   PIC X(13)
      VALUE IS 'VALUE="TCP4">'.
02 HIDDEN2.
   03 TYPE2                    PIC X(21)
      VALUE IS '<INPUT TYPE="hidden" '.
   03 NAME2                    PIC X(13)
      VALUE IS 'NAME="XAddr" '.
   03 VALUE2                   PIC X(20)
      VALUE IS 'VALUE="9.20.2.19">'.
02 HIDDEN3.
   03 TYPE3                    PIC X(21)
      VALUE IS '<INPUT TYPE="hidden" '.
   03 NAME3                    PIC X(13)
      VALUE IS 'NAME="XPort" '.
   03 VALUE3                   PIC X(13)
      VALUE IS 'VALUE="3456">'.
02 HR1                         PIC X(4)
   VALUE IS '<HR>'.
02 MSGTEXT1                    PIC X(22)
   VALUE IS 'Greetings from CICS<P>'.
02 MSGTEXT2.
   03 TEXT2-1                  PIC X(8)
   VALUE IS 'Action:'.
   03 TEXT2-2                  PIC X(23)
   VALUE IS '<select name="Action">'.
   03 TEXT2-3                  PIC X(14)
   VALUE IS '<OPTION>Read'.
   03 TEXT2-4                  PIC X(15)
   VALUE IS '<OPTION>Write'.
   03 TEXT2-5                  PIC X(16)
   VALUE IS '<OPTION>Delete'.
   03 TEXT2-6                  PIC X(15)
   VALUE IS '</select><p>'.
02 MSGTEXT3.
   03 TEXT3-1                  PIC X(31)
   VALUE IS 'Queue Name <INPUT TYPE="text"'.
   03 TEXT3-2                  PIC X(26)
   VALUE IS ' NAME="Queue" SIZE="10" '.
   03 TEXT3-3                  PIC X(20)
   VALUE IS ' MAXLENGTH="8"><P> '.
02 MSGTEXT4.
   03 TEXT4-1                  PIC X(31)
   VALUE IS 'Queue Data <INPUT TYPE="text"'.
   03 TEXT4-2                  PIC X(26)
   VALUE IS ' NAME="QData" SIZE="50" '.
   03 TEXT4-3                  PIC X(16)
   VALUE IS ' MAXLENGTH="50"'.
   03 TEXT4-4                  PIC X(8)
   VALUE IS ' VALUE="'.
   03 TSQUEUE-DATA             PIC X(50) VALUE IS '"'.
   03 TEXT4-5                  PIC X(8)
   VALUE IS '"><P>'.

02 HR2                         PIC X(8)
   VALUE IS ' <HR>  '.
02 BUTTON1.
   03 BUTT1-1                  PIC X(21)
   VALUE IS '<INPUT TYPE="submit" '.
   03 BUTT1-2                  PIC X(38)
   VALUE IS ' VALUE="Click here to send request">'.
02 BUTTON2.
   03 BUTT2-1                  PIC X(20)
   VALUE IS '<INPUT TYPE="reset" '.
   03 BUTT2-2                  PIC X(22)
```

```
       VALUE IS ' VALUE="Clear entry">'.
    02 FORM2                       PIC X(8)
       VALUE IS '</FORM>'.

*-------------------------------------------------------------*
*    program's variables                                      *
*-------------------------------------------------------------*
 77  TCP-TOKEN              PIC X(16) VALUE 'TCPIPIUCVSTREAMS'.
 77  TOEBCDIC-TOKEN         PIC X(16) VALUE 'TCPIPTOEBCDICXLT'.
 77  TOASCII-TOKEN          PIC X(16) VALUE 'TCPIPTOASCIIXLAT'.
 77  TAKE-SOCKET                  PIC 9(8) BINARY.
 77  SOCKID                       PIC 9(4) BINARY.
 77  SOCKID-FWD                   PIC 9(8) BINARY.
 77  TEMP-SOCK                    PIC 9(8) BINARY.
 77  ERRNO                        PIC 9(8) BINARY.
 77  RETCODE                      PIC S9(8) BINARY.
 77  AF-INET                      PIC 9(8) BINARY VALUE 2.
 77  TCP-BUF                      PIC X(2000) VALUE IS SPACES.
 77  TCPLENG                      PIC 9(8) BINARY.
 77  MAX-MSG-LEN                  PIC 9(8) BINARY VALUE 50.
 77  RECV-FLAG                    PIC 9(8) BINARY VALUE 0.
 77  CLENG                        PIC S9(4) COMP.

 77  QUEUE-DATA                   PIC X(50) VALUE IS SPACES.
 77  QUEUE-DATA-LEN               PIC S9(4) COMP VALUE 50.

 01  ZERO-PARM                PIC X(16) VALUE LOW-VALUES.
 01  DUMMY-MASK REDEFINES ZERO-PARM.
     05 DUMYMASK              PIC X(8).
     05 ZERO-FLD-8            PIC X(8).
 01  ZERO-FLD REDEFINES ZERO-PARM.
     05 ZERO-FWRD             PIC 9(8)  BINARY.
     05 ZERO-HWRD             PIC 9(4)  BINARY.
     05 ZERO-DUM              PIC X(10).

 77  CICS-MSG-AREA            PIC X(100).
 01  COMMAND.
     05 INITAPI-CMD           PIC S9(4) BINARY VALUE 0.
     05 CLOSE-CMD             PIC S9(4) BINARY VALUE 3.
     05 CONNECT-CMD           PIC S9(4) BINARY VALUE 4.
     05 FCNTL-CMD             PIC S9(4) BINARY VALUE 5.
     05 IOCTL-CMD             PIC S9(4) BINARY VALUE 12.
     05 READ-CMD              PIC S9(4) BINARY VALUE 14.
     05 RECV-CMD              PIC S9(4) BINARY VALUE 16.
     05 SEND-CMD              PIC S9(4) BINARY VALUE 20.
     05 SEND-TO-CMD           PIC S9(4) BINARY VALUE 22.
     05 WRITE-CMD             PIC S9(4) BINARY VALUE 26.
     05 GETCLIENT-ID-CMD      PIC S9(4) BINARY VALUE 30.
     05 GIVESOCKET-CMD        PIC S9(4) BINARY VALUE 31.
     05 TAKESOCKET-CMD        PIC S9(4) BINARY VALUE 32.
*
 01  CLIENTID-LSTN.
     05 CID-DOMAIN-LSTN             PIC 9(8) BINARY.
     05 CID-NAME-LSTN               PIC X(8).
     05 CID-SUBTASKNAME-LSTN        PIC X(8).
     05 CID-RES-LSTN                PIC X(20).

 01  CLIENTID-APPL.
     05 CID-DOMAIN-APPL             PIC 9(8) BINARY.
     05 CID-NAME-APPL               PIC X(8).
     05 CID-SUBTASKNAME-APPL        PIC X(8).
     05 CID-RES-APPL                PIC X(20).

 01  TCPSOCKET-PARM.
     05 GIVE-TAKE-SOCKET            PIC 9(8) BINARY.
     05 LSTN-NAME                   PIC X(8).
     05 LSTN-SUBTASKNAME            PIC X(8).
     05 CLIENT-IN-DATA              PIC X(100).
     05 SOCKADDR-IN.
       10  SIN-FAMILY             PIC 9(4).
       10  SIN-PORT               PIC 9(4).
       10  SIN-ADDR               PIC 9(8).
       10  SIN-ZERO               PIC X(8).


 PROCEDURE DIVISION.
```

```
     EXEC CICS HANDLE CONDITION INVREQ  (INVREQ-ERR-SEC)
                                IOERR   (IOERR-SEC)
                                ENDDATA (ENDDATA-SEC)
                                LENGERR (LENGERR-SEC)
                                NOSPACE (NOSPACE-ERR-SEC)
                                QIDERR  (QIDERR-SEC)
                                ITEMERR (ITEMERR-SEC)
     END-EXEC.

     PERFORM INITIAL-SEC     THRU   INITIAL-SEC-EXIT.
     PERFORM TAKESOCKET-SEC  THRU   TAKESOCKET-SEC-EXIT.

 CLOSE-SOCK.
*---------------------------------------------------------------*
*                                                               *
*   CLOSE 'accept descriptor'                                   *
*                                                               *
*---------------------------------------------------------------*

     CALL 'EZACICAL' USING TCP-TOKEN CLOSE-CMD SOCKID ZERO-FLD-8
          ERRNO RETCODE.

     IF RETCODE <  0 THEN
        MOVE CLOS-ERR TO CICS-MSG-AREA
     ELSE
        MOVE CLOS-SUCCESS TO CICS-MSG-AREA.
     PERFORM WRITE-CICS  THRU WRITE-CICS-EXIT.

 PGM-EXIT.

     MOVE SPACES TO CICS-MSG-AREA.
     MOVE 'END OF TCPSCOBC PROGRAM' TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     EXEC CICS RETURN  END-EXEC.

*---------------------------------------------------------------*
*
*  RECEIVE PASSED PARAMETERS                                    *
*
*---------------------------------------------------------------*
 INITIAL-SEC.

     MOVE SPACES TO CICS-MSG-AREA.
     MOVE LENGTH OF CICS-MSG-AREA TO CLENG.
     MOVE 'TCPC TRANSACTION START UP     ' TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

     MOVE LENGTH OF TCPSOCKET-PARM TO CLENG.

     EXEC CICS RETRIEVE INTO(TCPSOCKET-PARM) LENGTH(CLENG)
     END-EXEC.

     MOVE CLIENT-IN-DATA TO MSG-IN.

 INITIAL-SEC-EXIT.
     EXIT.

 RECV-CLIENT.
*---------------------------------------------------------------*
*                                                               *
*   Issue an acknowledgement back to the client - he's waiting  *
*   on this before he sends me more data.                       *
*                                                               *
*   Issue 'RECVFROM' call to receive more information from the  *
*   client program.                                             *
*                                                               *
*---------------------------------------------------------------*

     MOVE SPACES TO TCP-BUF.
     MOVE WRKMSG TO TCP-BUF.
     MOVE LENGTH OF TCP-BUF TO TCPLENG.

     CALL 'EZACIC04' USING TOASCII-TOKEN TCP-BUF TCPLENG.

     CALL 'EZACICAL' USING TCP-TOKEN WRITE-CMD SOCKID TCPLENG
          ZERO-FWRD ZERO-PARM TCP-BUF ERRNO RETCODE.

     IF RETCODE <  0 THEN
```

```
           MOVE WRITE-ERR TO CICS-MSG-AREA
           PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
           GO TO PGM-EXIT.

      * --- now to the receive

          MOVE LOW-VALUES TO TCP-BUF.
          MOVE MAX-MSG-LEN TO TCPLENG.
          CALL 'EZACICAL' USING TCP-TOKEN RECV-CMD SOCKID ZERO-FWRD
               RECV-FLAG TCPLENG SOCKADDR-IN TCP-BUF
               ERRNO RETCODE.

          IF RETCODE <  0 THEN
             MOVE READ-ERR TO CICS-MSG-AREA
             PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
             GO TO PGM-EXIT
          ELSE
              MOVE SPACES TO CICS-MSG-AREA
              MOVE READ-SUCCESS TO CICS-MSG-AREA
              PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

          CALL 'EZACIC05' USING TOEBCDIC-TOKEN TCP-BUF TCPLENG.

       RECV-CLIENT-EXIT.
          EXIT.

      *---------------------------------------------------------------*
      *                                                               *
      *  Perform TCP SOCKET functions by passing socket command to    *
      *  TCPRMCAL routine.  SOCKET command are translated to pre-     *
      *  define integer.                                              *
      *                                                               *
      *---------------------------------------------------------------*

       TAKESOCKET-SEC.
      *---------------------------------------------------------------*
      *                                                               *
      *   Issue 'TAKESOCKET' call (integer 32) to acquire a socket    *
      *   which was given by LISTERN program.                         *
      *                                                               *
      *---------------------------------------------------------------*

          MOVE AF-INET TO CID-DOMAIN-LSTN CID-DOMAIN-APPL.

          MOVE LSTN-NAME TO CID-NAME-LSTN.
          MOVE LSTN-SUBTASKNAME TO CID-SUBTASKNAME-LSTN.
          MOVE GIVE-TAKE-SOCKET TO TAKE-SOCKET SOCKID SOCKID-FWD.
          MOVE GIVE-TAKE-SOCKET TO TEMP-SOCK.

          CALL 'EZACICAL' USING TCP-TOKEN TAKESOCKET-CMD ZERO-HWRD
               CLIENTID-LSTN TAKE-SOCKET SOCKID-FWD
               ERRNO RETCODE.

          IF TAKE-SOCKET NOT EQUAL TEMP-SOCK THEN
             MOVE SOCK-DIFF TO CICS-MSG-AREA
             PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
             GO TO PGM-EXIT
          ELSE
             MOVE SPACES TO CICS-MSG-AREA
             MOVE SOCK-SAME TO CICS-MSG-AREA
             PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

          IF SOCKID-FWD NOT EQUAL TEMP-SOCK THEN
             MOVE SOCK-DIFF TO CICS-MSG-AREA
             PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
             GO TO PGM-EXIT
          ELSE
             MOVE SPACES TO CICS-MSG-AREA
             MOVE SOCK-SAME TO CICS-MSG-AREA
             PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

          IF RETCODE <  0 THEN
             MOVE TAKE-ERR TO CICS-MSG-AREA
             PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
             GO TO PGM-EXIT
          ELSE
              MOVE SPACES TO CICS-MSG-AREA
              MOVE TAKE-SUCCESS TO CICS-MSG-AREA
```

```
        PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

*
* WE MUST LOOK AT THE DATA FROM THE CLIENT, AND DECIDE
* WHETHER TO:
*
*      A) SEND BACK INITIAL PANEL, OR
*      B) READ QUEUE DATA AND RETURN DATA, OR
*      C) WRITE QUEUE DATA TO SPECIFIED TS QUEUE, OR
*      D) DELETE THE SPECIFIED QUEUE.
*
*
* SO THE FIRST PART OF THE 'IF' CLAUSE IS EXECUTED INITIALLY,
* AND THE SECOND PART EXECUTED WHEN A QUEUEID IS SPECIFIED
*

     IF QUEUE-ID = SPACES THEN
         MOVE FIRST-REQUEST TO CICS-MSG-AREA
         PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
     ELSE
         IF QUEUE-ACTION = 'R' THEN
             MOVE MSG-IN TO CICS-MSG-AREA
             PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
             PERFORM READ-QUEUE THRU READ-QUEUE-EXIT
         ELSE
             IF QUEUE-ACTION = 'W' THEN
                 MOVE MSG-IN TO CICS-MSG-AREA
                 PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
                 PERFORM RECV-CLIENT THRU RECV-CLIENT-EXIT
                 MOVE TCP-BUF TO CICS-MSG-AREA
                 PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
                 PERFORM WRITE-QUEUE THRU WRITE-QUEUE-EXIT
             ELSE
                 MOVE MSG-IN TO CICS-MSG-AREA
                 PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
                 PERFORM DELETE-QUEUE THRU DELETE-QUEUE-EXIT.


     MOVE SPACES TO TCP-BUF.
     MOVE NEW-TASK-START TO TCP-BUF.
     MOVE LENGTH OF TCP-BUF TO TCPLENG.

     MOVE TCP-BUF TO CICS-MSG-AREA
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT

     CALL 'EZACIC04' USING TOASCII-TOKEN TCP-BUF TCPLENG.

     CALL 'EZACICAL' USING TCP-TOKEN WRITE-CMD SOCKID TCPLENG
          ZERO-FWRD ZERO-PARM TCP-BUF ERRNO RETCODE.

     IF RETCODE <  0 THEN
        MOVE WRITE-ERR TO CICS-MSG-AREA
        PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
        GO TO PGM-EXIT.

 TAKESOCKET-SEC-EXIT.
     EXIT.


*---------------------------------------------------------------*
*   READ QUEUE FOR CLIENT AND SEND HIM THE INFO.
*---------------------------------------------------------------*
 READ-QUEUE.

     EXEC CICS READQ TS QUEUE(QUEUE-ID)
          INTO(QUEUE-DATA) LENGTH(QUEUE-DATA-LEN)
          ITEM(1)
     END-EXEC.

     MOVE QUEUE-DATA TO TSQUEUE-DATA.
     MOVE TSQUEUE-DATA TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

 READ-QUEUE-EXIT.
     EXIT.


*---------------------------------------------------------------*
```

```
*   WRITE QUEUE DATA FOR CLIENT
*---------------------------------------------------------------*
 WRITE-QUEUE.

     EXEC CICS WRITEQ TS QUEUE(QUEUE-ID)
          FROM(TCP-BUF) LENGTH(QUEUE-DATA-LEN)
     END-EXEC.

 WRITE-QUEUE-EXIT.
     EXIT.

*---------------------------------------------------------------*
*   DELETE QUEUE FOR CLIENT
*---------------------------------------------------------------*
 DELETE-QUEUE.

     EXEC CICS DELETEQ TS QUEUE(QUEUE-ID)
     END-EXEC.

 DELETE-QUEUE-EXIT.
     EXIT.


*---------------------------------------------------------------*
*   WRITE OUT TO CSMT LOG
*---------------------------------------------------------------*
 WRITE-CICS.
     MOVE LENGTH OF CICS-MSG-AREA TO CLENG.
     EXEC CICS WRITEQ TD QUEUE('CSMT') FROM(CICS-MSG-AREA)
          LENGTH(CLENG) NOHANDLE
     END-EXEC.

     MOVE SPACES TO CICS-MSG-AREA.

 WRITE-CICS-EXIT.
     EXIT.

 INVREQ-ERR-SEC.
     MOVE INVREQ-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
 IOERR-SEC.
     MOVE IOERR-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
 LENGERR-SEC.
     MOVE LENGERR-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
 NOSPACE-ERR-SEC.
     MOVE NOSPACE-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
 QIDERR-SEC.
     MOVE QIDERR-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
 ITEMERR-SEC.
     MOVE ITEMERR-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
 ENDDATA-SEC.
     MOVE ENDDATA-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
```

D.3 MVS JCL to Compile COBOL Program

```
//CICSRS2C JOB (999,POK),'CICSRS2',NOTIFY=CICSRS2,
//    CLASS=A,MSGCLASS=T,TIME=1439,
//    REGION=5000K,MSGLEVEL=(1,1)
//DFHEITVL PROC SUFFIX=1c,
//       INDEX='CICS330',
//       INDEX2='CICS330',
//       OUTC=*,
```

```
//       REG=2048K,
//       LNKPARM='LIST,XREF',
//       WORK=SYSDA
//TRN    EXEC PGM=DFHECP&SUFFIX,
//            PARM='COBOL2',
//            REGION=&REG
//STEPLIB  DD DSN=&INDEX2..SDFHLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSPUNCH DD DSN=&&SYSCIN,
//            DISP=(,PASS),UNIT=&WORK,
//            DCB=BLKSIZE=400,
//            SPACE=(400,(400,100))
//*
//COB    EXEC PGM=IGYCRCTL,REGION=&REG,
//       PARM='NODYNAM,LIB,OBJECT,RENT,RES,APOST,MAP,XREF'
//STEPLIB  DD DSN=COBOL.V1R3M2.COB2COMP,DISP=SHR
//SYSLIB   DD DSN=&INDEX..SDFHCOB,DISP=SHR
//         DD DSN=&INDEX..SDFHMAC,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSIN    DD DSN=&&SYSCIN,DISP=(OLD,DELETE)
//SYSLIN   DD DSN=&&LOADSET,DISP=(MOD,PASS),
//            UNIT=&WORK,SPACE=(80,(250,100))
//SYSUT1   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT2   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT3   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT4   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT5   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT6   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT7   DD UNIT=&WORK,SPACE=(460,(350,100))
//*
//LKED   EXEC PGM=IEWL,REGION=&REG,
//            PARM='&LNKPARM',COND=(5,LT,COB)
//SYSLIB   DD DSN=&INDEX..SDFHLOAD,DISP=SHR
//         DD DSN=SYS1.COBOL.V1R3M2.COB2CICS,DISP=SHR
//         DD DSN=COBOL.V1R3M2.COB2LIB,DISP=SHR
//         DD DSN=TCPIP.V2R2M1.SEZATCP,DISP=SHR
//SYSLMOD  DD DSN=CICSRS2.CICS330.PGMLIB,DISP=SHR
//SYSUT1   DD UNIT=&WORK,DCB=BLKSIZE=1024,
//            SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=&OUTC
//SYSLIN   DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//         DD DDNAME=SYSIN
//   PEND
//APPLPROG EXEC DFHEITVL
//TRN.SYSIN  DD DISP=SHR,DSN=CICSRS2.JCL.DATA(TCPSERV1)
//LKED.SYSIN DD *
   NAME TCPSERV1(R)
/*
```

Appendix E. Special Notices

This publication is intended to help information technology professionals responsible for designing and managing CICS-based Online Transaction Processing (OLTP) applications, to help them make these applications available to users of the Internet and the World Wide Web. The information in this publication is not intended as the specification of any programming interfaces that are provided by the products mentioned in this book. See the PUBLICATIONS section of the IBM Programming Announcement for these products for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	AIX/6000
AIXwindows	BookManager
BookMaster	C Set++
CICS	CICS OS/2
CICS for Windows/NT	CICS for OS/2
CICS/ESA	CICS/MVS
CICS/VSE	CICS/400
CICS/6000	DB2
DB2/2	Enterprise Systems Architecture/390
ESA/390	IBM
MQSeries	MVS/ESA
OS/2	Presentation Manager
RACF	RISC System/6000
RS/6000	S/390
System/390	VTAM

The following terms are trademarks of other companies:

C++	American Telephone and Telegraph Company, Incorporated
C-bus	Corollary, Inc.
CyberCash	CyberCash, Inc.
DCE, OSF	Open Software Foundation
DEC, VT100	Digital Equipment Corporation
Encina	Transarc Corporation
Gopher	University of Minnesota
Intel, Pentium, MMX	Intel Corporation
Java, Hot Java	Sun Microsystems, Inc.
Kerberos, X Windows	Massachusetts Institute of Technology
Lotus Notes	Lotus Development Corporation.
NCSA Mosaic	University of Illinois at Urbana Champaign
Netscape	Netscape Communications Corporation
Oracle	Oracle Corporation
PostScript	Adobe Systems, Inc.
SecureWeb	Terisa Systems
Windows NT, and Windows 95	Microsoft Corporation

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Appendix F. Related Publications

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

F.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How To Get ITSO Redbooks” on page 235.

- *IBM Internet Connection Secure Server*, SG24-4805
- *Building a Firewall With the NetSP Secured Network Gateway*, SG24-2577
- *CICS Clients Unmasked*, SG24-2534
- *CICS/ESA and TCP/IP for MVS Sockets Interface*, GG24-4026
- *TCP/IP Tutorial and Technical Overview*, GG24-3376-04
- *Using the Information Super Highway*, SG24-2499

F.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

F.3 Other Publications

These publications are also relevant as further information sources:

- *CICS/ESA Dynamic Transaction Routing in a CICSplex*, SC33-1012
- *CICS Family: Client/Server Programming*, SC33-1435
- *CICS Family: Communicating from CICS on System/390, SC33-1697*
- *CICS Internet and External Interfaces Guide*, SC33-1944
- *Internet Cryptography*, ISBN 0-20192-480-3
- *Priciples of Transaction Processing*, ISBN 1-55860-415-4
- *Web Gateway Tools,* ISBN 0-47117-555-2

How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at `http://www.redbooks.ibm.com`.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** – to order hardcopies in United States
- **GOPHER link to the Internet** – type `GOPHER WTSCPOK.ITSO.IBM.COM`
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Web Site on the World Wide Web**

  `http://w3.itso.ibm.com/redbooks`

- **IBM Direct Publications Catalog on the World Wide Web**

  `http://www.elink.ibmlink.ibm.com/pbl/pbl`

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** – send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) – send orders to:

	IBMMAIL	**Internet**
In United States	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** – send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** – send orders to:

United States (toll free)	1-800-445-9269
Canada	1-800-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 408 256 5422 (Outside USA)** – ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** – send note to softwareshop@vnet.ibm.com

- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.html). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name | Last name

Company

Address

City | Postal code | Country

Telephone number | Telefax number | VAT number

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date | Card issued to | Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

There is an excellent glossary of Internet and Internet related terms available at URL:

http://www.matisse.net/files/glossary.html

Other terms not covered in the above-mentioned Web document or clarified for the context of this document are listed below:

Anchor. An HTML element that defines a link between Internet resources.

abend. Abnormal end of task.

API. Application programming interface. A set of calling conventions defining how a service is invoked through a software package.

APPC. Advanced program-to-program communication. An implementation of the SNA LU 6.2 protocol that allows interconnected systems to communicate and share the processing of programs.

asynchronous. Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions. See *synchronous*.

Browser. An application that displays World Wide Web documents.

CERN. The Conseil Europeen pour la Recherche Nucleaire (European Particle Physics Laboratory), which developed hypertext technologies.

Distributed program link (DPL) enables an application program executing in one CICS system to link (pass control) to a program in a different CICS system. The linked-to program executes and returns a result to the linking program. This process is equivalent to remote procedure calls (RPCs). You can write applications that issue RPCs that can be received by members of the CICS family.

Distributed transaction processing enables a transaction running in one CICS system to communicate synchronously with transactions running in other systems. The transactions are designed and coded specifically to communicate with each other. This method is typically used by banks, for example in "just-in-time" stock replacement.

Customer Information Control System (CICS). A distributed online transaction processing system designed to support a network of many terminals. The CICS family of products is available for a variety of platforms ranging from a single workstation to the largest mainframe.

client. As in client/server computing, the application that makes requests to the server and, often, handles with the interaction necessary with the user.

client/server computing. A form of distributed processing, in which the task required to be processed is accomplished by a client portion that requests services and a server portion that fulfills those requests. The client and server remain transparent to each other in terms of location and platform. See *client* and *server*.

commit. An action that an application takes to make permanent the changes it has made to CICS resources.

Common Gateway Interface (CGI). The defined standard for the communications between HTTP servers and external executable programs.

conversational. A communication model where two distributed applications exchange information by way of a conversation; typically one application starts (or allocates) the conversation, sends some data, and allows the other application to send some data. Both applications continue in turn until one decides to finish (or deallocate). The conversational model is a synchronous form of communication.

database. (1) A collection of interrelated data stored together with controlled redundancy according to a scheme to serve one or more applications. (2) All data files stored in the system. (3) A set of data stored together and managed by a database management system.

DCE. Distributed Computing Environment. Adopted by the computer industry as a de facto standard for distributed computing. DCE allows computers from a variety of vendors to communicate transparently and share resources such as computing power, files, printers, and other objects in the network.

Delimiter. A character or sequence of characters used as a separator in text or data files.

distributed processing. Distributed processing is an application or systems model in which function and data can be distributed across multiple computing resources connected on a LAN or WAN. See *client/server computing*.

DPL. Distributed program link. Provides a mechanism similar to remote procedure call (RPC) by function shipping EXEC CICS LINK commands. In CICS/6000, DPL allows the linked-to program to issue unsupported CICS/6000 function-shipping calls, such as to DB2 and DL/I, and yields performance improvements for transactions that read many records from remote files.

DTP. Distributed transaction processing. A type of intercommunication in CICS. The processing is distributed between transactions that communicate synchronously with one another over intersystem links. DTP enables a CICS application program to initiate transaction processing in a system that supports LU 6.2 and resides in the same or a different processor.

ECI. External call interface. An application programming interface (API) that enables a non-CICS

client application to call a CICS program as a subroutine. The client application communicates with the server CICS program using a data area called a COMMAREA.

EPI. External presentation interface. An application programming interface (API) that allows a non-CICS application program to appear to the CICS system as one or more standard 3270 terminals. The non-CICS application can start CICS transactions and send and receive standard 3270 data streams to those transactions.

environment. The collective hardware and software configuration of a system.

File Transfer Protocol (FTP). A protocol that defines how to transfer files from one computer to another.

Forms. Parts of HTML documents that allow users to enter data.

Function shipping enables an application program running in one CICS system to access resources owned by another CICS system. In the resource-owning system, a transaction is initiated to perform the necessary operation; for example, to access CICS files or temporary storage, and to reply to the requester. The user is unaware of these "behind-the-scenes" activities, and need not know where the resource actually exists.

Gateway. Software that transfers data between normally incompatible applications or between networks.

Gopher. Menu-based software for exploring Internet resources.

Graphic Interchange Format (GIF). 256-color graphic format.

GUI. Graphical user interface. A style of user interface that replaces the character-based screen with an all-points-addressable, high-resolution graphics screen. Windows display multiple applications at the same time and allow user input by means of a keyboard or a pointing device such as mouse, a pen, or a trackball.

home page. The default page shown at the first connection to an HTTP server.

host. (1) In a computer network, a computer providing services such as computation, database access, and network control functions. (2) In a multiple computer installation, the primary or controlling computer.

hypertext. Text that activates connection to other documents when selected.

Hypertext Markup Language (HTML). Standard language used to create hypertext documents.

Hypertext Transmission Protocol (HTTP). Standard WWW client/server communications protocol.

Internet Keyed Payment Protocol (iKP). Proposed protocol for conducting secure commercial financial transactions on the Internet.

intercommunication. Communication between separate systems by means of Systems Network Architecture (SNA), Transmission Control Protocol/Internet Protocol (TCP/IP), and Network Basic Input/Output System (NetBIOS) networking facilities.

Internet. A collection of networks.

LU type 6.2 (LU 6.2). A type of logical unit used for CICS intersystem communication (ISC). LU 6.2 architecture supports CICS host-to-system-level products and CICS host-to-device-level products. APPC is the protocol boundary of the LU 6.2 architecture.

LUW. Logical unit of work. An update that durably transforms a resource from one consistent state to another consistent state. A sequence of processing actions (for example, database changes) that must be completed before any of the individual actions can be regarded as committed. When changes are committed (by successful completion of the LUW and recording of the synch point on the system log), they do not need to be backed out after a subsequent error within the task or region. The end of an LUW is marked in a transaction by a synch point that is issued by either the user program or the CICS server, at the end of task. If there are no user synch points, the entire task is an LUW.

markup tag. Special character sequences put in text used to pass information to a tool, such as a document formatter.

NCSA Mosaic. A Web browser available on multiple platforms.

Multipurpose Internet Mail Extension (MIME). The Internet standard for mail that supports text, images, audio, and video.

online transaction processing (OLTP). A style of computing that supports interactive applications in which requests submitted by terminal users are processed as soon as they are received. Results are returned to the requester in a relatively short period of time. An online transaction processing system supervises the sharing of resources to allow efficient processing of multiple transactions at the same time.

PostScript. The standard for presenting text and graphics in a device-independent format.

protocol. (1) A formal set of conventions governing the format and control of data. (2) A set of procedures or rules for establishing and controlling transmissions from a source device or process to a target device or process.

Proxy. A gateway that allows Web browsers to pass on a network request (a URL) to an outside agent.

pseudoconversational. A type of CICS application design that appears to the user as a continuous conversation but consists internally of multiple tasks.

recovery. The use of archived copies to reconstruct files, databases, or complete disk images after they are lost or destroyed.

recoverable resources. Items whose integrity CICS maintains in the event of a system error. These include individual files and queues.

script. An executable program invoked by HTTP servers.

server. Any computing resource dedicated to responding to client requests. Servers can be linked to clients through LANs or WANs to perform services, such as printing, database access, fax, and image processing, on behalf of multiple clients at the same time.

Socket Secure (SOCKS). The gateway that allows compliant client code (client code made socket secure) to establish a session with a remote host.

Standard Generalized Markup Language (SGML). The standard that defines several markup languages, HTML included

synchronous. (1) Pertaining to two or more processes that depend on the occurrence of a specific event such as a common timing signal. (2) Occurring with a regular or predictable time relationship.

synchpoint. A logical point in execution of an application program where the changes made to the databases by the program are consistent and complete and can be committed to the database. The output, which has been held up to that point, is sent to its destination, the input is removed from the message queues, and the database updates are made available to other applications. When a program terminates abnormally, CICS recovery and restart facilities do not back out updates prior to the last completed synchpoint.

transaction. A unit of processing (consisting of one or more application programs) initiated by a single request. A transaction can require the initiation of one or more tasks for its execution.

transaction processing. A style of computing that supports interactive applications in which requests submitted by users are processed as soon as they are received. Results are returned to the requester in a relatively short period of time. A transaction processing system supervises the sharing of resources for processing multiple transactions at the same time.

Transaction routing enables a terminal connected to one CICS system to run a transaction in another CICS system. It is common for CICS/ESA, CICS/VSE, and CICS/MVS users to have a terminal-owning region (TOR) that "owns" end-user network resources, an application-owning region (AOR) that owns user transactions and programs, and a resource-owning region (ROR) that owns data resources. These resources can include files, temporary storage queues, and transient data queues, and may have access to database managers such as DBCTL or DB2.

Uniform Resource Locator (URL). Standard to identify resources on the World Wide Web

WebExplorer. OS/2 Web browser.

workstation. A configuration of input/output equipment at which an operator works. A terminal or microcomputer, usually one that is connected to a mainframe or a network, at which a user can perform applications.

World Wide Web (WWW or W3). A graphic hypertextual multimedia Internet service.

X-Windows Systems. Network-based windowing system originally developed by the Massachusetts Institute of Technology (MIT).

List of Abbreviations

ACF	access control file
ACL	access control list
AIX	Advanced Interacrtive eXecutive
APA	all points addressable
API	application programming interface
APPC	Advanced Program-to-Program Communication
ASCII	American National Standard Code for Information Interchange
BMS	basic mapping support
CERN	Conseil Europeen pour la Recherche Nucleaire (European Laboratory for Particle Physics)
CGI	Common Gateway Interface
CICS	Customer Information Control System
CM/2	Communications Manager/2
COMMAREA	communication area
CSD	CICS system definition
DCE	Distributed Computing Environment
DEC	Digital Equipment Corporation
DNS	Domain Name Server
DOS	Disk Operating System
DPL	distributed program link
DTP	distributed transaction processing
ECI	external call interface
EPI	external presentation interface
ESA	Enterprise Systems Architecture
EXCI	external CICS interface
FAT	file allocation table
FTP	File Transfer Protocol
GIF	graphic interchange format
HPFS	High Performance File System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IETF	Internet Engineering Task Force
iKP	Internet Keyed Payment Protocol
IP	Internet Protocol
ISC	intersystem communication
ITSO	International Technical Support Organization
LAN	local area network
LUW	logical unit of work
MIME	Multipurpose Internet Mail Extension
NCSA	National Center of Supercomputing Applications
OLTP	online transaction processing
ONC RPC	Open Network Computing Remote Procedure Call
OS/2	Operating System/2
OSF	Open Software Foundation Inc.
PGP	Pretty Good Privacy
PIN	personal identification number
PM	Presentation Manager
POP	Post Office Protocol
RACF	Resource Access Control Facility
RPC	remote procedure call
SET	Secure Electronic Transaction
S-HTTP	Secure Hypertext Transfer Protocol
SGML	Standard Generalized Markup Language
SMTP	Simple Mail Transfer Protocol
SNA	Systems Network Architecture
SNT	sign-on table
SOCKS	socket secure
SSL	Secure Socket Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TOR	terminal owning region

TRUE	task-related user exit
URI	Uniform Resource Identifier
URL	Uniform Resource Locator or Universal Resource Locator
WWW	World Wide Web
WYSIWYG	What-you-see-is-what-you-get
W3	World Wide Web (mostly used for Intranet declaration).

Index

Numerics

A

C

D

E

F

G

H

T

U

W

X

ITSO Redbook Evaluation

Accessing CICS Business Applications from the World Wide Web
SG24-4547-02

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@vnet.ibm.com

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____

_____

_____

_____

_____

SG24-4547-02

Printed in the U.S.A.

Accessing CICS Business Applications from the World Wide Web

SG24-4547-02