CICS® Transaction Gateway for OS/390®

Administration

Version 3.1

SC34-5528-01

CICS® Transaction Gateway for OS/390®

Administration

Version 3.1

SC34-5528-01

Note!
Before using this information and the product it supports, be sure to read the general information under "Appendix B. Notices" on page 93.

Second edition (September 1999)

This edition applies to Version 3.1 of CICS Transaction Gateway for OS/390 program number 5648-B43. It will also apply to all subsequent versions, releases, and modifications until otherwise indicated in new editions.

This edition replaces SC34-5528-00.

Contents

Figures

Tables

About this book

This book contains the following chapters:

- Chapter 1 introduces the CICS Transaction Gateway for OS/390 and summarizes the benefits of using it, and the functions it provides.
- Chapter 2 discusses planning issues including the software requirements of CICS Transaction Gateway for OS/390, and various migration issues.
- Chapter 3 describes how to install your CICS Transaction Gateway for OS/390.
- Chapter 4 describes how to configure the CICS Transaction Gateway for OS/390.
- Chapter 5 describes how to set up security for the SSL and HTTPS protocols in the CICS Transaction Gateway for OS/390.
- Chapter 6 describes how to operate the CICS Transaction Gateway for OS/390, including starting and stopping the CICS Transaction Gateway for OS/390.
- Chapter 7 provides an introduction to programming with the CICS Transaction Gateway for OS/390.
- Chapter 8 describes problem determination for CICS Transaction Gateway for OS/390.
- Appendix A describes how to view the online information in the CICS Transaction Gateway library.

Who should read this book

This book is intended for anyone involved with the planning, installation, customization, operation, or programming of a CICS Transaction Gateway for OS/390.

It is assumed that you are familiar with the OS/390 operating system.

An understanding of Internet terminology would also be helpful.

Prerequisite and related information

The following sections list books relevant to CICS Transaction Gateway for OS/390.

IBM CICS Transaction Server for OS/390 and other publications

CICS Internet Guide, SC34-5445

CICS External Interfaces Guide, SC33-1944

CICS Problem Determination Guide, GC33-1693

CICS Resource Definition Guide, SC33-1684

CICS System Definition Guide, SC33-1682

CICS Family: Communicating from CICS on System/390®, GC33-1693

IBM CICS Transaction Gateway related publications

For information on the books available for this product, refer to "Appendix A. The CICS Transaction Gateway and CICS Universal Clients library" on page 87. That chapter also gives details of how to view and print softcopy books, and how to order printed copies from IBM.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book, or any other CICS documentation:

- Visit our Web site at:

  `http://www.ibm.com/software/ts/cics/`

  and follow the **Library** link to our feedback form.

  Here you will find the feedback page where you can enter and submit your comments.
- Send your comments by e-mail to idrcf@hursley.ibm.com
- Fax your comments to:

  +44-1962-870229 (if you are outside the UK)
  01962-870229 (if you are in the UK)
- Mail your comments to:

  Information Development
  Mail Point 095
  IBM United Kingdom Laboratories
  Hursley Park
  Winchester
  Hampshire
  SO21 2JN
  United Kingdom

Whichever method you use, ensure that you include:

- The name of the book
- The form number of the book
- If applicable, the version of the product
- The specific location of the text you are commenting on, for example, a page number or table number.

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.

Obtaining books from IBM

For information on books you can download, visit our Web site at:

`http://www.ibm.com/software/ts/cics/`

and follow the **Library** link.

You can order hardcopy books:

- Through your IBM representative or the IBM branch office serving your locality.
- By calling 1-800-879-2755 in the United States.
- From the Web site at:

  `http://www.elink.ibmlink.ibm.com/pbl/pbl`

Chapter 1. CICS Transaction Gateway overview

The IBM CICS Transaction Gateway provides secure, easy access from Web browsers and network computers to business-critical applications running on a CICS Transaction Server or TXSeries™ server using standard Internet protocols in a range of configurations.

The CICS Transaction Gateway is provided for the OS/2®, Windows NT®, AIX,® and Solaris platforms. The CICS Transaction Gateway is also provided for Windows® 95/98, but on these platforms it can only be used for development and not for production purposes.

CICS Transaction Gateway for OS/390 is provided for the OS/390 platform. However, unlike the other CICS Transaction Gateways, which can access multiple CICS servers, CICS Transaction Gateway for OS/390 can only access Transaction Server for OS/390.

The CICS Transaction Gateway for OS/2, Windows, AIX, and Solaris use a CICS Universal Client to route external call interface (ECI), external presentation interface (EPI), and external security interface (ESI) requests to a CICS server, see “The external access interfaces (EPI, ECI, and ESI)” on page 11. On the other hand, CICS Transaction Gateway for OS/390 can only route ECI requests and not EPI or ESI requests. The CICS Transaction Gateway for OS/390 actually uses the CICS External Call Interface (EXCI) to pass requests to CICS, see “CICS External Call Interface (EXCI)” on page 13. However, to a Java™ applet or application these appear to be ECI requests.

Figure 1 on page 2 shows how a Web-client can access CICS programs and data. Note that the CICS Transaction Gateway is shown as installed on a Web server machine. CICS Transaction Gateway need only be installed on the Web server machine if you are using the CICS Transaction Gateway with Java applets.

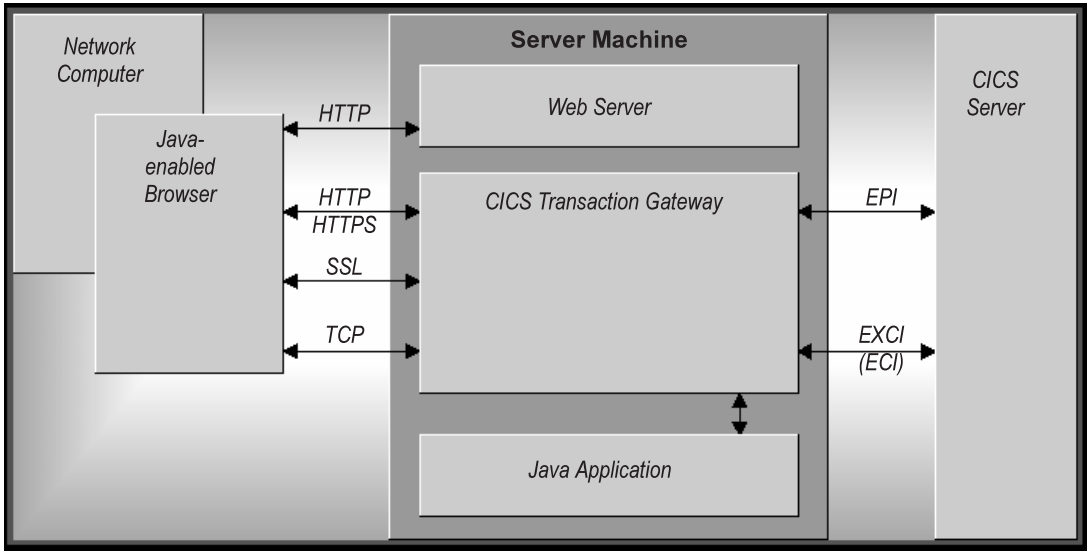

Figure 1. CICS Transaction Gateway

Communication with the CICS Transaction Gateway is based on the following protocols:

- TCP/IP sockets
- Hypertext Transfer Protocol (HTTP)
- Secure Sockets Layer (SSL)
- HTTP over SSL (HTTPS)

TCP/IP sockets and SSL provide an efficient method of communication for intranet applications. Where firewalls exist, HTTP and its secure alternative HTTPS, are effective communication protocols for Internet applications (see "Network security" on page 13).

What the CICS Transaction Gateway for OS/390 provides

The CICS Transaction Gateway for OS/390 provides the following:

1. A **Java gateway application** that communicates with CICS applications running in CICS servers through the EXCI (External CICS Interface). This Java application was previously available in the IBM CICS Gateway for Java (MVS).
2. A **CICS Java class library** that includes classes that provide an application programming interface (API), and are used to communicate between the Java gateway application and a Java application (applet). The class **JavaGateway** is used to establish communication with the Gateway

process, and uses the Java sockets protocol. The class **ECIRequest** is used to specify the ECI calls that are flowed to the gateway. These Java classes were previously available in the IBM CICS Gateway for Java (MVS).

The CICS Transaction Gateway for OS/390 can concurrently manage many communication links to connected Web browsers, The multithreaded architecture of the CICS Transaction Gateway for OS/390 enables a single Gateway to support multiple concurrently connected users.

Java technology

This section discusses the Java language, including the types of program that can be developed, and the security implications.

The Java language

The Java language can be used to construct Java *applets* and Java *applications.*

Java is an interpreted object-oriented language, similar to C++ that can be used to build programs that are platform-independent in both source and object form. Its unique operational characteristics, which span Web browsers as well as Web servers, enable new and powerful functions in Internet applications.

To achieve platform independence, the Java language allows no platform dependent-operations, and it excludes some C++ functions such as a preprocessor, operator overloading, multiple inheritances, and pointers. All Java programming is encapsulated within classes, and the Java Development Kit (JDK) includes special classes that are critical to assuring platform independence, and includes GUI functions, input/output functions, and network communications.

The Java compiler produces an intermediate bytecode format that is machine independent. This, in turn, is processed at execution time by a Java interpreter. The interpreter also inspects the bytecode at execution time to ensure its validity and safety to the machine environment. Because of the isolation the Java interpreter provides, it is sometimes referred to as a Java Virtual Machine.

Java applets

A Java applet is a small application program that is downloaded to, and executed on, a Java-enabled Web browser or network computer. A Java applet typically performs the type of operations that client code would perform in a client/server architecture. It edits input, controls the screen, and sends transactions to a server that performs data or database operations.

Applets are started using the <applet> HTML tag; this gives the applet control and specifies 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 bytecode, in the same way that it downloads an image that is referenced by an HTML image tag. The Java-enabled browser then interprets and executes the applet bytecode. The applet may edit screen input, generate screen output, and communicate back to the computer from which it was downloaded. Multiple applets can execute concurrently.

An example of applet processing is an applet in constant communication with a server to receive stock trade information, which it would update in a window on the screen.

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. Applets may even improve overall browser performance by eliminating iterations with the Web server. Also, just as images are cached in Web browsers, applets are cached, minimizing the frequency of applet downloading.

Java applications

A Java application is a program that executes locally on a computer. It has platform-dependent capabilities in addition to those of an applet. It can access local files, create and accept general network connections, and call native C or C++ functions in machine-specific libraries.

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

- A *network* gateway connection is a connection across a network to a CICS Transaction Gateway.
- A *local* gateway enables a Java application to communicate directly with a locally-installed CICS Transaction Gateway for OS/390, without the need for a network.

When the connection between the application and the CICS Transaction Gateway for OS/390 has been established, an application can use the **ECIRequest** class to do transaction processing. (For CICS Transaction Gateway for OS/390, the **EPIRequest** and **ESIRequest** classes cannot be used.)

Firewalls

A current design consideration in the use of Java applet communication is the impact of *firewalls*. This is the term given to a configuration of software that

prevents unauthorized traffic between a trusted network and an untrusted network. Firewalls are put in place to protect company assets from outside intrusion, but they can also limit legitimate communications as well. Firewalls come into play in two ways:

1. The general accessibility of a server to outside users - inbound restrictions
2. The ability of end users inside a firewall to perform certain network functions outside their firewall - outbound restrictions.

A CICS Transaction Gateway for OS/390 configuration is well suited to avoid problems in the first area since the Gateway processor can be placed outside the firewall and be connected through the firewall to the CICS server. Outbound firewalls that end users may have to contend with can be a problem. A large company might use a firewall to limit the types of connections and protocols that can be used.

The use of Java on an Intranet (a local implementation of the Internet) works very well since firewalls are typically not a factor. However, when designing Internet applications for end users outside a company, you should determine if end user firewalls will be an implementation factor. If so, then alternative processing for those users, such as executing the Java code as a Java servlet on the Web server, may be necessary. Also you should consider the use of the HTTP and HTTPS protocols supported by CICS Transaction Gateway for OS/390, see “Secure Sockets Layer (SSL)” on page 13 and “HTTPS” on page 15.

Web browsers and network computers

The CICS Transaction Gateway for OS/390 requires a Java-enabled Web browser, that is JDK version 1.1 enabled, for example, Netscape Navigator 2.0.2 or later on OS/2, Netscape Communicator 4.5.1 or later on Windows and AIX. For more information, refer to “Web browsers” on page 17.

The Web browser communicates with the Web server using HyperText Transport Protocol (HTTP) requesting HyperText Markup Language (HTML) pages to be downloaded. These HTML pages can include calls to Java applets (see “Java applets” on page 3); multiple applets can run concurrently.

You may find that each browser displays the same information differently.

A network computer is a low-cost computer for the Internet user, it does the same things as a Web browser.

Web servers

A Web server is a software program that responds to information requests generated by Web browsers. When a request from a browser is received, the Web server processes the request to determine the action to take:

- Return the requested document
- Deny the request
- Pass the request through for further processing by an external application. The request might be, for example, to a database to perform a search request, or to a more dynamic form of information delivery such as Lotus Domino™.

Communication between a Web server and an external application is transparent, you need to know only the URL of the Web server to direct a request to it. Also, all Web servers can handle requests from many browsers concurrently.

Specialized servers can also be configured to limit access to a restricted set of users, or to provide security for purchase of goods or services.

Web servers exist for almost every platform and are available from many suppliers. For information on the Web servers supported by CICS Transaction Gateway for OS/390, see “Web servers” on page 17.

How the CICS Transaction Gateway for OS/390 accesses CICS

This section describes how the CICS Transaction Gateway for OS/390 allows access to CICS programs and data.

Figure 2 on page 7 shows the flow of control when a Web browser calls CICS transaction processing facilities using the CICS Transaction Gateway for OS/390.

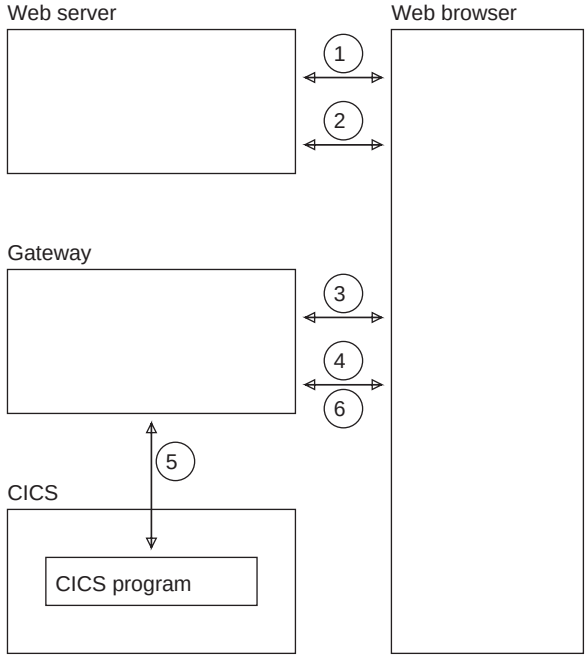

Figure 2. Flow of control with the CICS Transaction Gateway for OS/390

1. The Web browser calls the Web server using HTTP (Hypertext Transfer Protocol) to get HTML pages.
2. When the browser, which is interpreting the HTML and presenting it on the screen, finds an applet tag, it calls the Web server to get the applet and the classes that it needs. It then executes the applet.
3. An applet that is going to communicate with CICS creates a **JavaGateway** object. The creation of this object causes a call to the CICS Transaction Gateway for OS/390 long-running task.
4. The applet creates an **ECIRequest** object to represent its request for a CICS program, and calls the **flow** method of the **JavaGateway** object, passing the instance of the **ECIRequest** object.
5. The CICS Transaction Gateway for OS/390 receives the request, and calls the CICS program.
6. When the CICS program ends, the results are returned to the Web browser.

Figure 3 on page 8 shows the flow of data when a Web browser calls CICS transaction processing facilities using the gateway.

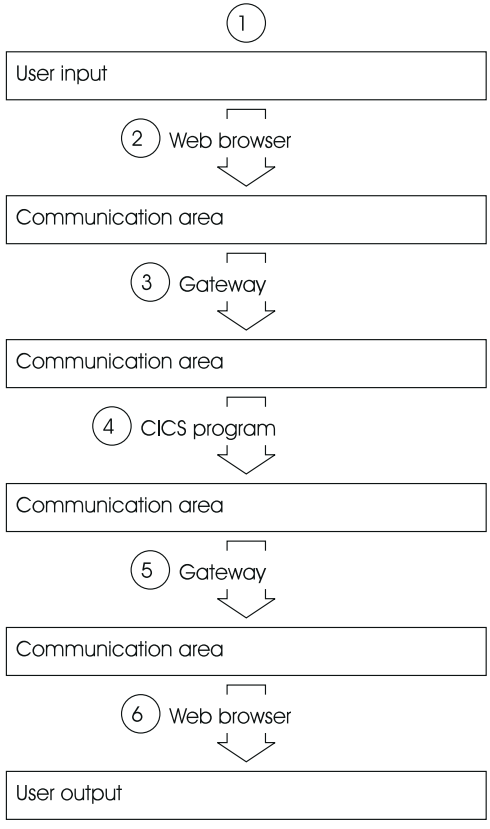

Figure 3. Data flow with the CICS Transaction Gateway for OS/390

1. The Web browser acquires data for the CICS program from the user.
2. The Web browser constructs a communication area (COMMAREA) for the CICS program that is to supply transaction processing services.
3. The CICS Transaction Gateway for OS/390 receives the communication area and passes it to the CICS program. The contents of the communication area are translated from ASCII (in the gateway) to EBCDIC (in the CICS Transaction Server).
4. The CICS program supplies the transaction processing services, enquiring on and perhaps changing CICS resources. If the program ends normally, changes to recoverable resources are committed. If the program ends abnormally, the changes are backed out.
5. The communication area is translated from EBCDIC to ASCII, and returned by the gateway to the Web browser.
6. The Web browser presents information to the user.

The CICS Transaction Gateway: threading model

The CICS Transaction Gateway provides a multithreaded model for handling network connections, and assigning threads for requests from / replies to Java clients. The following components are involved in the threading model:

ConnectionManagers

A ConnectionManager manages all the connections from a particular Java client (applet or application). When it receives a request, it allocates a Worker thread from a pool of available Worker threads to execute the request. The size of the initial ConnectionManager resource pool is defined by the **Initial number of Connection Manager threads** configuration setting. You can specify the maximum size of the ConnectionManager pool by using the **Maximum number of Connection Manager threads** setting, (see "Using the configuration tool" on page 27). You can also specify these limits when you start the CICS Transaction Gateway, (see "Starting from a command line" on page 71).

Workers

A Worker is the object that actually executes a request from a Java client. Each Worker object has its own thread, which is activated when there is some work to do. When a worker thread is finished it goes back into the pool of available worker threads. As with the ConnectionManager, the Worker resource pool has a initial size that is specified using the **Initial number of Worker threads** setting. You can specify the maximum size of the Worker pool with the **Maximum number of Worker threads** setting, see "Using the configuration tool" on page 27. You can also specify these limits when you start the CICS Transaction Gateway, (see "Starting from a command line" on page 71).

The threading model is illustrated in the following figures:

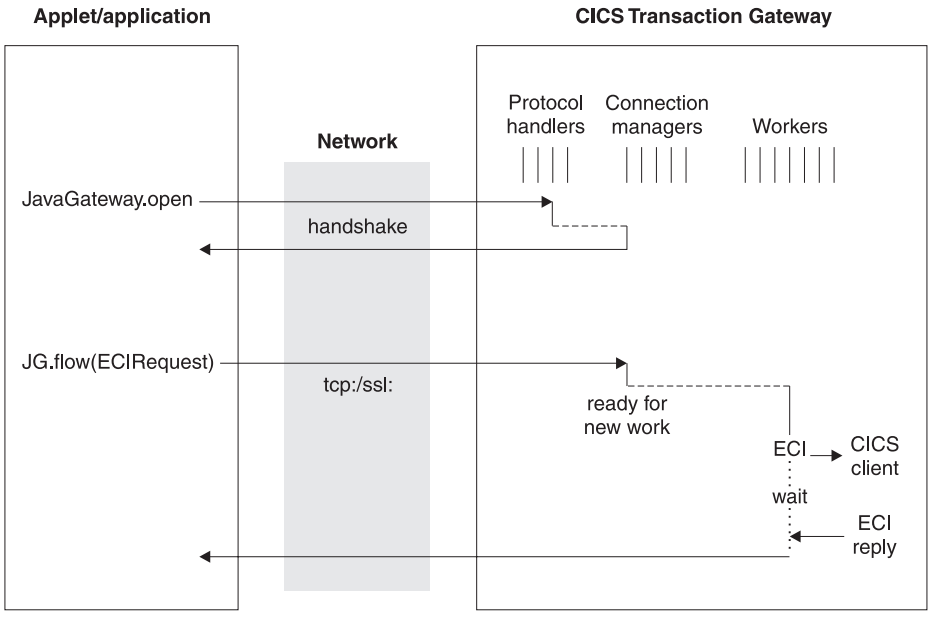

Figure 4. CICS Transaction Gateway threading model for tcp/ssl

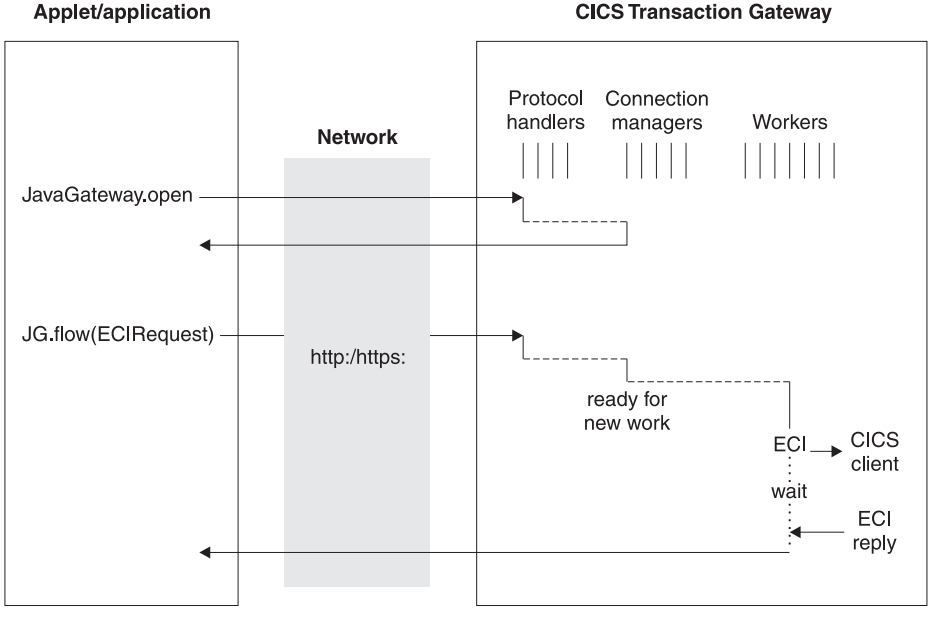

Figure 5. CICS Transaction Gateway threading model for http/https

Note: For CICS Transaction Gateway for OS/390 the ECI calls shown in the figures are mapped to EXCI calls.

Table 1 shows thread limits that should be considered when setting the number of Connection Manager and Worker threads on the various platforms:

Table 1. Thread limits on CICS Transaction Gateway platforms

Platform	System-wide limit of the maximum number of threads	Process limit of the number of threads
OS/390	This may be restricted by the total number of MVS TCBs (one is created per OpenEdition thread)	Governed by the OpenEdition parameters: MAXTHREADS and MAXTHREADTASKS
OS/2	4095	Governed by the THREAD parameter in config.sys
Windows 95/98/NT	No limit	Limited by the amount of virtual memory available for the process (by default a thread has 1M of stack meaning that 2028 threads can be created per process)
AIX	262,143	32768
Solaris	No limit	No limit

You can set the stack size of the Java threads using the Java -oss and -ss options. Note that the amount of memory allocated per thread can be the limiting factor, because memory can run out before the thread limit is reached.

For more information on the use of the Java -oss and -ss options, refer to the *CICS Transaction Gateway Programming* book.

The external access interfaces (EPI, ECI, and ESI)

The external access interfaces allow non-CICS applications to access and update CICS resources by initiating CICS transactions, or by calling CICS programs. When used in conjunction with the CICS communication facilities, they enable non-CICS programs to access and update resources on any CICS system. This supports such activities as: developing graphical user interface (GUI) front ends for CICS applications, and allowing the integration of CICS systems and non-CICS systems.

The external presentation interface (EPI) allows you to develop GUIs, either for existing CICS systems or for new applications. It is particularly useful for developing new GUI front ends for existing CICS transactions, which need

not be changed. The application can use the EPI to communicate with a CICS transaction, and can exploit the presentation facilities of the client system to communicate with the end user.

The external security interface (ESI) allows a non-CICS application to invoke services provided by advanced program-to-program communication (APPC) password expiration management (PEM).

The external presentation interface (EPI) and external security interface (ESI) are **not** supported by the CICS Transaction Gateway for OS/390.

The integration of CICS and non-CICS systems usually involves passing user-defined data between the programs of the non-CICS system and a CICS program, and the external call interface (ECI) can be used for this.

External Call Interface (ECI)

The ECI allows a non-CICS application to call a CICS program in a CICS server. The application can be connected to several servers at the same time, and it can have several program calls outstanding at the same time.

The CICS program cannot perform terminal I/O, but can access and update all other CICS resources. The same CICS program can be called by a non-CICS application using the ECI, or by a CICS program using EXEC CICS LINK. Data is exchanged between the two programs by means of a COMMAREA, in a similar way to CICS. The user can specify the length of the COMMAREA data to optimize performance.

Calls may be made synchronously or asynchronously. Synchronous calls return control when the called program completes, and the information returned is immediately available. Asynchronous calls return control without reference to the completion of the called program, and the application can ask to be notified when the information becomes available.

Calls may also be *extended*. That is, a single logical unit of work may cover two or more successive calls, though only one call can be active for each logical unit of work at any time. If it uses asynchronous calls, the application can manage many logical units of work concurrently.

The called program can update resources on its own system, it can use distributed program link (DPL) to call CICS programs on other systems, and it can access resources on other CICS systems by function shipping, by distributed transaction processing (DTP), or (in the Transaction Server for OS/390™ environment) by the front end programming interface (FEPI).

For more information on the external access interfaces, see *CICS Family: Client/Server Programming*.

CICS External Call Interface (EXCI)

The external CICS interface (EXCI) is analogous to the ECI, and is an application programming interface that enables a program running in MVS, such as the CICS Transaction Gateway for OS/390, to call a program running in a CICS region and to pass and receive data by means of a communication area.

The EXCI allows a user to allocate and open sessions (or pipes) to a CICS region, and to pass distributed program link (DPL) requests over them. The multiregion operation (MRO) facility of CICS inter-region communication (IRC) facility supports these requests, and each pipe maps onto one MRO session, with a limit of 100 pipes per EXCI address space.

For more information on the EXCI, refer to the *CICS External Interfaces Guide*.

Network security

The CICS Transaction Gateway supports the use of the Secure Sockets Layer (SSL) and HTTPS protocols to provide secure communication, which is critical to successful Internet operation.

Network security and its implementation on CICS Transaction Gateway is discussed in detail in "Chapter 5. CICS Transaction Gateway for OS/390 security" on page 39; the following sections summarize the functions provided by SSL and HTTPS.

Secure Sockets Layer (SSL)

SSL is a **Handshake Protocol** developed to provide security and privacy over the Internet. The use of the SSL protocol ensures:

Confidentiality

The data to be exchanged between the client and the server is encrypted, so only that client (your application or applet) and that server (the CICS Transaction Gateway) can make sense of the data.

SSL uses public key encryption as a secure mechanism to distribute a secret key between the server and the client. Public key encryption is a technique that uses a pair of

asymmetric keys for encryption and decryption. In the case of SSL, a secret (symmetric) key is passed between the client and server (using public key cryptography), which is then used to encrypt and decrypt all traffic along the SSL connection. This encryption protects the data from other parties trying to eavesdrop, as no other parties will have the secret key needed to decrypt the data. This ensures that private information such as a credit card number is transferred securely.

Integrity The message transport includes a message integrity check based on a secure hashing algorithm. This algorithm is performed when the message is sent, and again when it is received. If the two hash values do not match, the receiver is warned that the message may have been tampered with.

Accountability

Accountability is ensured by digital signature, so that if something goes wrong, you can identify which party is accountable.

Authentication

The CICS Transaction Gateway's implementation of the SSL protocol provides *server authentication*, so that when a client establishes a connection with the CICS Transaction Gateway, it is required to authenticate the server's details. *Client authentication* can also be enabled, in which case the server will authenticate the client's details,

The authentication mechanism is based on the exchange of digital certificates (X.509v3 certificates). These digital certificates contain information about an entity, like the system name and public key, and the server's digital signature. Digital certificates are issued by a Certificate Authority (CA), and encrypted using the CA's private key. If you can decrypt the certificate using the CA's public key, you know that the information contained within the certificate can be trusted, that is, that the certificate really does belong to whoever claims to own it.

System SSL and SSLight

Two types of SSL are supported for CICS Transaction Gateway for OS/390:

System SSL is a platform-specific implementation of the SSL protocol, that is written in native code and which supports hardware cryptographic technology available to OS/390.

SSLight is a pure Java implementation of SSL.

The client-side code (that is, the code running the distributed applet), will always use SSLight. However, on the server side, System SSL should be used in preference to SSLight, because it will provide a significant performance improvement.

HTTPS

Most current browsers support a URL access method, HTTPS, for connecting to HTTP servers using SSL. HTTPS (HTTP+SSL) is a variant of HTTP for handling secure transactions.

A secure connection is typically made with a URL similar to "https://someAddress" The default HTTPS port number is 443, as assigned by the Internet Assigned Numbers Authority.

Keys and Certificates

The SSL protocol uses public key cryptography, which has been recommended for use with the ISO authentication framework, also known as the X.509 protocol. This framework provides for authentication across networks.

The most important part of X.509 is its structure for public key certificates. A trusted Certification Authority (CA) assigns a unique name to each user and issues a signed certificate containing the name and the user's public key.

The CICS Transaction Gateway allows you to obtain externally-signed certificates from a CA, or to establish yourself as a CA to allow you to issue ″self-signed″ X.509 certificates. Externally-signed certificates are more suitable for Internet use, while self-signed certificates may be suitable for internal use within an organization.

For System SSL, the X.509 digital certificates are "encapsulated" into a database file, (.kdb), and for SSLight into a Java classfile, and these can then be used by the SSL and HTTPS protocols. The database files or Java classfiles are referred to as *KeyRings*.

For System SSL, the CICS Transaction Gateway uses the MVS system tool, GSKKYMAN for management of digital certificates. For SSLight, the iKeyMan tool is included with CICS Transaction Gateway for the same purpose. Using these tools, you can create KeyRing files, generate certificates, export certificates, and perform various other management functions. See "Chapter 5. CICS Transaction Gateway for OS/390 security" on page 39 for more information.

Security exits

Security Exits are provided that enable the user to define security operations such as public key encryption. They may also be used for data compression. Some example source files that demonstrate these functions are provided.

It is also possible for the CICS Transaction Gateway to expose Client Certificates in server-side security exits. These in turn can be mapped to RACF userids, see “Mapping client certificates to RACF userids” on page 65 for more information.

For more information, see the *CICS Transaction Gateway Programming* book.

Chapter 2. CICS Transaction Gateway for OS/390 planning

This chapter helps you plan the installation of the CICS Transaction Gateway for OS/390 by discussing the software requirements, the Web Servers and browsers that are supported, and also migration issues.

Software requirements

For the development of Java applets and applications, **Java Development Kit (JDK)** Version 1.1.8 is required.

Note: JIT compilers may cause problems with the HTTPS and SSL protocols and therefore should be disabled if you wish to use these secure protocols.

The following levels of OS/390 are required:

- OS/390 V2R5.0, or later for OpenEdition® support
- OS/390 V2R6.0, or later for Transactional EXCI support (RRS) support
- OS/390 V2R7.0, or later for System SSL support.

Web servers

CICS Transaction Gateway for OS/390 supports the following Web servers:

- Lotus Domino Go Webserver™
- WebSphere® Advanced Server for OS/390

Web browsers

The CICS Transaction Gateway for OS/390 should work with any JDK Version 1.1 compliant Java-enabled browser. This includes the following:

- OS/2: Netscape 2.0.2 with 1.1 patch
- Windows NT, 98, 95: Microsoft Internet Explorer Version 4.0.1, or later (Note that HTTPS as transport does not work.)
- Windows NT, 98, 95: Netscape Version 4.0.8, Netscape Communicator Version 4.51
- AIX: Netscape Version 4.0.8, Netscape Communicator Version 4.51
- Solaris: HotJava™ Browser Version 1.1

You can also use the JDK AppletViewer to run applets.

Restrictions

There is a compatibility problem between Internet Explorer Version 4 (or later) and the CICS Transaction Gateway's HTTPS protocol. This means that it is not possible to establish a https: connection to a CICS Transaction Gateway from an applet running within Internet Explorer's Java Virtual Machine.

Migrating from CICS Gateway for Java (MVS)

If you are a user of CICS Gateway for Java (MVS) migrating to CICS Transaction Gateway for OS/390, you may have customized the Gateways.properties or the JGate script to set environment variables. If you have customized these files, you should save a copy of them before you install CICS Transaction Gateway for OS/390.

In CICS Transaction Gateway for OS/390 Version 3.1, the Gateway.properties file has been renamed to CTG.INI and the JGate script to ctgstart. You can use the **ctgconv** utility to convert old Gateway.properties files to the new format.

Due to the renaming of the CICS Transaction Gateway for OS/390 package, you must change all `import` statements in your programs and recompile them. Therefore you must change all occurrences of:

`import ibm.cics.jgate.client.*` to: `import com.ibm.ctg.client.*`

`import ibm.cics.jgate.security.*` to: `import com.ibm.ctg.security.*`

Due to a change in the ClientSecurity and ServerSecurity interfaces, any user classes that implement these methods need to be changed. The methods called to generate handshake data are now passed the TCP/IP address of who they are handshaking with. Also, an afterDecode method has been added to both interfaces.

Migrating from CICS Transaction Gateway for OS/390 Version 3.0

You may have customized the JGate script to set environment variables. If you have customized this file, you should save a copy before you install CICS Transaction Gateway for OS/390 Version 3.1.

In Version 3.1, the Gateway.properties file has been renamed to CTG.INI, and the JGate script to ctgstart. You can use the **ctgconv** utility to convert old Gateway.properties files to the new format, see "The configuration conversion tool" on page 34.

Note: You can use the configuration tool on the other CICS Transaction Gateway platforms to generate a CTG.INI file for use on the OS/390

platform. For more information, see the the *CICS Transaction Gateway Administration* books for the other platforms.

Chapter 3. Installing CICS Transaction Gateway for OS/390

This chapter describes how to install CICS Transaction Gateway for OS/390.

Installation

The CICS Transaction Gateway for OS/390 is provided as the compressed file CTG-310m.tar.Z, which contains the CICS Transaction Gateway for OS/390 software that is to be installed in the hierarchical file system (HFS).

This material must first be copied into a directory on a workstation that can communicate with the System/390 system on which you wish to install the CICS Transaction Gateway for OS/390.

When you have copied the program material to the workstation, perform the following actions:

1. Transfer the program material from the workstation into a suitable directory in HFS as follows:
   a. Upload the file to an MVS sequential data set. You can use your terminal emulator transfer options to do this.
   b. Use the OPUT command on TSO to put the data set in HFS, for example:

      ```
      OPUT 'CTG-310M.TAR.Z' '/path/CTG-310m.tar.Z' BINARY
      ```

      where *path* is the HFS path that is to be the root for the CICS Transaction Gateway for OS/390 directory structure.
2. Use the OMVS command on TSO to enter the OS/390 UNIX® System Services shell. Then use the command

   ```
   cd /path
   ```

   to change to the directory into which you put CTG-310m.tar.Z.
3. Use the command

   ```
   uncompress CTG-310m.tar.Z
   ```

   to uncompress the file.
4. Use the command

   ```
   tar -xopfv CTG-310m.tar
   ```

   where:

- `-x` specifies that all files are extracted
- `-o` allows the original file permission bits for owner, group and all to be restored. This is useful for all the execute bits.
- `-p` does not try to restore the original owner and group id when extracting files. The default is to do this, but most likely the person doing the extract will not have permission, or the user id will not exist on the system.
- `-f` Specifies the file name of the archive file.
- `-v` specifies verbose messages (this is optional).

This creates the appropriate subdirectories.

Note: To transfer the program material to HFS you can also use other methods, for example, using ftp.

CICS definitions for the CICS Transaction Gateway for OS/390

The CICS Transaction Gateway for OS/390 needs a session definition and a connection definition for EXCI. For details of how to produce these definitions see the *CICS External Interfaces Guide*. The following are examples:

```
VIEW GROUP(MYEXCI) CONNECTION(JCOS)
 OBJECT CHARACTERISTICS                                  CICS RELEASE = 0530
  CEDA  View Connection( JCOS )
   Connection     : JCOS
   Group          : MYEXCI
   DEscription    : Sample EXCI Specific connection
  CONNECTION IDENTIFIERS
   Netname        : JGATE310
   INDsys         :
  REMOTE ATTRIBUTES
   REMOTESYSTem   :
   REMOTEName     :
   REMOTESYSNet   :
  CONNECTION PROPERTIES
   ACcessmethod   : IRc               Vtam | IRc | INdirect | Xm
   PRotocol       : Exci              Appc | Lu61 | Exci
   Conntype       : Specific          Generic | Specific
   SInglesess     : No                No | Yes
   DAtastream     : User              User | 3270 | SCs | STrfield | Lms
+  RECordformat   : U                 U | Vb

                                                  SYSID=CW2C APPLID=IYCWZCFY

PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 6. Example connection definition

```
VIEW GROUP(MYEXCI) SESSIONS(JCOS)
 OBJECT CHARACTERISTICS                                  CICS RELEASE = 0530
  CEDA  View Sessions( JCOS     )
   Sessions       : JCOS
   Group          : MYEXCI
   DEscription    : Sample EXCI Specific sessions definition
  SESSION IDENTIFIERS
   Connection     : JCOS
   SESSName       :
   NETnameq       :
   MOdename       :
  SESSION PROPERTIES
   Protocol       : Exci              Appc | Lu61 | Exci
   MAximum        : 000 , 000         0-999
   RECEIVEPfx     : JG
   RECEIVECount   : 004               1-999
   SENDPfx        :
   SENDCount      :                   1-999
   SENDSize       : 04096             1-30720
+  RECEIVESize    : 04096             1-30720

                                                SYSID=CW2C APPLID=IYCWZCFY

PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 7. Example sessions definition

The above examples define a specific connection for the CICS Transaction Gateway to use. The value specified to the CICS Transaction Gateway in the environment variable DFHJVPIPE must be the same as the **Netname** option of the connection definition (shown in the figure as **JGATE310**). The netname itself is an arbitrary value. A generic pipe is defined in much the same way, except that the **Netname** option is left blank.

Note: If you use the sample DFH$EXCI, you must set the DFHJVPIPE environment variable to BATCHCLI in order to use the specific pipe. If you do not set the variable, the generic pipe is used. For more information see “Environment variables” on page 25.

Using X-Windows from a remote system

When using X-Windows from a remote system, for example, to access the configuration tool and iKeyman, you must set up the DISPLAY environment variable to allow the application to display its windows on that system.

On the display system (that is the one that will display the windows), enter the command:

```
xhost +appl
```

where *appl* is the network name of the system being used to run the application.

On the application system, before you run the application, enter the command:

```
DISPLAY=disp:0
```

followed by

```
export DISPLAY
```

where *disp* is the network name of the system where the windows will be displayed (followed by a colon and the display id—normally 0). The application windows are then displayed on the *disp* system.

You can add the statement:

```
export DISPLAY=disp:0
```

to your .profile file.

Chapter 4. Configuring CICS Transaction Gateway for OS/390

This chapter describes what you can do to configure CICS Transaction Gateway for OS/390:

- Set properties of the CICS Transaction Gateway for OS/390 in the CTG.INI file.
- Use environment variables, for example, to control the use of EXCI pipes, and the names of CICS servers that callers can access.
- Edit the ctgstart script that is used to start CICS Transaction Gateway for OS/390.

Environment variables

You can set the following environment variables for CICS Transaction Gateway for OS/390:

AUTH_USERID_PASSWORD
Specifies whether the userid and password passed on an EXCI call should be authenticated with RACF. To disable this feature, uncomment the following line in the ctgstart script:

```
#unset AUTH_USERID_PASSWORD
```

CLASSPATH
The path in HFS for the CICS-provided Java class definitions, that is, /ctg/classes. You **must** set CLASSPATH to compile and run Java applications. You should concatenate the existing CLASSPATH to the new CLASSPATH, for example:

```
export CLASSPATH=/u/joe/ctg/classes/ctgclient.jar:/u/joe/ctg/classes/ctgserver.jar:$CLASSPATH
```

DFHJVPIPE
The name of the specific pipe that the CICS Transaction Gateway for OS/390 is to use for EXCI calls. If this variable is not set, the CICS Transaction Gateway for OS/390 uses a generic pipe.

DFHJVSYSTEM_*nn*
You may set up to 100 environment variables of the form DFHJVSYSTEM_*nn*, where *nn* ranges from 00 to 99. The first variable must be DFHJVSYSTEM_00, and subsequent variables must have consecutive numbers, or they will not be recognized. The value of the variable is the name and description of a CICS system to be returned in response to a request for the **CICS_EciListSystems** function. The

value must be a string containing the uppercase APPLID of the CICS system immediately after the equals sign (=), followed by a hyphen (-), followed by a description of the CICS system, which must not be more than 60 bytes long. For example if you specify:

```
DFHJVSYSTEM_00=MYCICS-Test CICS system
```

CICS_EciListSystems returns details of a CICS system called MYCICS with description `Test CICS system`.

LIBPATH

The path in HFS for the CICS Transaction Gateway for OS/390 executable code. You must set this variable to `/ctg/bin`. You should concatenate the existing LIBPATH to the new LIBPATH:

```
export LIBPATH=/ctg/bin:$LIBPATH
```

RRM_NAME

The name of the resource manager managing this instance of the CICS Transaction Gateway for OS/390. For Resource Recovery Services, the name must be unique across a sysplex.

The name can consist of the following printable characters:

1. Alphanumeric characters: A-Z and 0-9
2. National characters: $ (X'5B'), # (X'7B'), @ (X'7C')
3. The period (.)
4. The underscore (_)

The name cannot start with a blank or contain embedded blanks. Lower case characters are converted to upper case characters.

To avoid naming conflicts, use A-C or G-I as the first character. The length of CTG_RRMNAME must not exceed 32 characters.

```
RRM_NAME="CCLTST.CTGTG"

export CTG_RRMNAME="${RRM_NAME}.IBM.UA"
```

STEPLIB

The library containing the default EXCI options and the EXCI load modules. This must be the same as specified for EXCI_LOADLIB in the ctgstart script, see “Customizing the ctgstart script” on page 35.

You can set the environment variables in three ways:

1. By entering commands on an OS/390 UNIX® System Services command line. For example, the following command, issued in the /ctg/bin directory, sets the DFHJVPIPE environment variable to use a specific pipe called JAVAGAT1.

   ```
   export DFHJVPIPE=JAVAGAT1
   ```

If the value of an environment variable contains spaces, it must be enclosed in single or double quotes. For example:

```
export DFHJVSYSTEM_00="MYCICS-Test CICS system"
```

2. By editing the ctgstart script that is used to start the gateway. See "Customizing the ctgstart script" on page 35.
3. By creating JCL to start the gateway that includes environment variable settings. See "Starting with JCL" on page 72.

Using the configuration tool

You use the configuration tool to set configuration parameters for the CICS Transaction Gateway for OS/390

To use the configuration tool, you must export the display using the commands described in "Using X-Windows from a remote system" on page 23.

To start the configuration tool, enter the **ctgcfg** command.

The configuration is stored by default in the CTG.INI file in the bin subdirectory where you installed the CICS Transaction Gateway. You can edit this file directly, but it is recommended that you use the configuration tool to perform configuration.

The configuration file contains equivalent entries to the Gateway.properties file of CICS Transaction Gateway Version 3.0.

If a configuration file already exists when you start the configuration tool, the settings in the file are loaded into the configuration tool.

The configuration tool interface

The user interface of the configuration tool consists of a menu bar, toolbar, tree structure, and Settings panel.

Tree structure

The tree structure allows you to navigate through all of the settings in your configuration. Under the **Gateway** node there are up to six subnodes, that is, one for each protocol the CICS Transaction Gateway can use.

Menu bar

The menu bar contains the **File, Tools,** and **Help** pulldowns.

The **File** pulldown has the following options:

New Creates a new configuration.

Open Opens an existing configuration.

Save Saves the current configuration. If a new configuration has been created, the file name is CTG.INI (in the bin subdirectory).

Save As Allows you to override the default path and name of the configuration file.

Exit Exits the configuration tool. You are asked whether you want to save the configuration.

The **Tools** pulldown has the following option:

Trace Settings Displays the Trace Settings dialog.

The **Help** pulldown has the following options:

Context Help Displays online help information according to the current screen context, for example, for a particular configuration setting.

Contents Displays the contents list for the online help information.

Index Displays a subject index for the online help information.

Toolbar

The toolbar has the same functionality as that of the menu bar, except it does not include the **Save as** or **Exit** options. The icons have hover helps, so that when you move the cursor over them, a text box describing the option appears.

Settings panels

When you select nodes in the tree structure, the relevant settings panel is displayed. The settings in these panels correspond to parameters in the CTG.INI file.

On each settings panel there is an **Undo Changes** button that allows you to undo changes you have made.

Configuring CICS Transaction Gateway settings

Using the configuration tool, you can provide preset values for any parameter that can be specified using a Gateway command line option.

If a parameter defined using the configuration tool is specified via the associated command line option when the Gateway is started, the command line setting ***takes precedence***.

General Gateway settings

The general Gateway settings are:

Initial number of Connection Manager threads: Enter the initial number of ConnectionManager threads. The default is 10.

You can override this setting with the `ctgstart -initconnect` command.

Maximum number of Connection Manager threads: Enter the maximum number of ConnectionManager threads. The default is 100.

If this value is set to -1, no limits are applied to the number of ConnectionManager threads.

You can override this setting with the `ctgstart -maxconnect` command.

For information on threading limits, see Table 1 on page 11.

Initial number of Worker threads: Enter the initial number of Worker threads. The default is 20.

You can override this setting with the `ctgstart -initworker` command.

Maximum number of Worker threads: Enter the maximum number of Worker threads. The default is 200.

If this value is set to -1, no limits are applied to the number of Worker threads.

You can override this setting with the `ctgstart -maxworker` command.

For information on threading limits, see Table 1 on page 11.

Time shown in messages: Select this check box to enable timing information in messages. (Timings are shown to millisecond accuracy.)

Timing information is enabled by default.

You can override this setting with the `ctgstart -notime` command.

Enable reading input from console: Select this check box to enable the reading of input from the console.

Reading of input from the console is enabled by default.

You can override this setting with the `ctgstart -noinput` command.

Display TCP/IP hostnames: Select this check box to enable the display of TCP/IP hostnames.

You can override this setting with the `ctgstart -nonames` command.

Let Java clients obtain generic ECI replies: Select this check box if you wish Java clients to be able to obtain generic ECI replies from the CICS Transaction Gateway.

Generic replies are those obtained using the Call_Type: ECI_GET_REPLY or ECI_GET_REPLY_WAIT. Specific replies are those obtained using the Call_Type: ECI_GET_SPECIFIC_REPLY or ECI_GET_SPECIFIC_REPLY_WAIT. This setting does not apply to local Gateways.

Timeout for in-progress requests to complete: Enter a value in milliseconds.

When a Java-client program disconnects from the Gateway, the Gateway may still be processing requests on behalf of that program. If work is still in progress, the ConnectionManager that was managing requests on behalf of that Java-client waits for in-progress requests to complete for up to the timeout period. If after this period there are still requests in progress, the ConnectionManager continues its processing and marks itself as available for use by a new connection. By default this timeout is set to 1000 milliseconds, but you may enter a value to override that default.

If this value is set to zero, the ConnectionManager does not wait for in-progress requests to complete.

Worker thread available timeout: Enter a value in milliseconds.

When a ConnectionManager accepts a request, it must allocate a Worker thread to execute that request. If however, a Worker does not become available within the timeout period an error message is sent rejecting that request and the request is not executed. By default, this timeout is set to 1000 milliseconds, but you can enter a value to override that default.

If this value is set to zero, the request is rejected, if a Worker is not immediately available.

TCP protocol settings

Select the **TCP** subnode to display the settings for TCP:

Enable protocol handler: Select this check box to configure the CICS Transaction Gateway for using this protocol.

Port: Enter the TCP/IP port number for the protocol:

For TCP, the default is 2006.

For HTTP, the default is 8080.

For System SSL, the default is 8050.

For SSL, the default is 8050.

For System HTTPS, the default is 443.

For HTTPS, the default is 443.

You can override this setting for the TCP protocol with the `ctgstart -port` command.

Handler wakeup timeout: Enter a value in milliseconds.

This setting controls how frequently the protocol handler wakes from accepting inbound connections. When it wakes, it checks to see whether the Gateway is being stopped, and so this value affects the time taken for the Gateway to shutdown cleanly. If you set this value to zero then the handler only wakes when a new connection is accepted, and so the Gateway will not shutdown cleanly until that time.

Connection timeout: Enter a value in milliseconds.

When a new connection has been accepted, this value specifies how long the protocol handler waits for a ConnectionManager thread to become available. If a ConnectionManager thread does not become available, then the connection is refused. If this value is set to zero, a connection is refused if a ConnectionManager is not immediately available.

Idle timeout: Enter a value in milliseconds.

This setting specifies how long a connection is allowed to remain dormant. The idle timeout period is counted from when a request was last flowed down the connection. When the idle timeout has expired, the connection is disconnected, though if work is still in progress on behalf of the connection, it may be left connected. If this value is not set, or is set to zero, then idle connections will not be disconnected.

Drop working connections: Select this check box to specify that a connection can be disconnected, due to an idle timeout or a PING/PONG failure ***even*** if work is still in progress on behalf of this connection.

SO_LINGER setting: Enter a SO_LINGER setting for any Socket used by this handler. If this value is not entered, or set to zero, then SO_LINGER is disabled for any Sockets used by this protocol handler.

Ping time frequency: Enter a value in milliseconds.

This value specifies how often a PING message is sent by the Gateway to an attached client to check that client is still active. If a PONG response has not been received by the time the next PING message is due to be sent, then the connection is disconnected. Again, if work is still in progress on behalf of the connection it may be left connected. If this value is not set, or is set to zero, then PING messages are not sent.

System SSL protocol settings

Select the **System SSL** subnode to display the settings for System SSL. The settings are the same as for TCP, with the following in addition:

Require security: Select this check box if you wish your Gateway to only accept connections that use security classes.

When a Java-client program connects to the Gateway, it can specify a pair of security classes that should be used on the connection. However, by default a Gateway also accepts connections from programs that do not specify this pair of security classes.

You can control which security classes are valid by controlling the set of *xxx*ServerSecurity classes that can be accessed by your Gateway.

For more information on CICS Transaction Gateway security exits, see “Security exits” on page 16.

KeyRing database: Enter the database file name (.kdb) of the server KeyRing.

The server KeyRing should consist of a valid x.509 certificate that is used to identify this server to connecting clients. This KeyRing database is generated using the System SSL tool (GSKKYMAN) supplied with this product.

For more information about SSL and its associated KeyRing databases, see "Chapter 5. CICS Transaction Gateway for OS/390 security" on page 39.

KeyRing password: Enter the Password for the server KeyRing class you specified during the creation process.

Use client authentication: Select this check box to enable client authentication for the CICS Transaction Gateway. The default is that client authentication is disabled.

When client authentication is enabled, any connection attempted to either the ssl: or https: handler requires the client to present its own Client Certificate (also known as a Digital ID).

For information on how to obtain Client Certificates for the clients, see "Configuring SSL clients" on page 50.

The default port for System SSL is 8050.

SSL protocol settings

Select the **SSL** subnode to display the settings for SSLight. The settings are the same as for System SSL, except for the following setting, which is used instead of **KeyRing database**.

KeyRing classname: Enter the class name of the server KeyRing.

The server KeyRing should consist of a valid x.509 certificate that is used to identify this server to connecting clients. This KeyRing class is generated using the SSL tools supplied with this product.

For more information about SSL and its associated KeyRing classes, see "Chapter 5. CICS Transaction Gateway for OS/390 security" on page 39.

The default port for SSL is 8050.

HTTP protocol settings

Select the **HTTP** subnode to display the settings for HTTP. The settings are the same as for TCP, except that there is no Ping time frequency setting.

The default port for HTTP is 8080.

System SSL HTTPS protocol settings

Select the **System SSL HTTPS** subnode to display the settings for HTTPS over System SSL. The settings are the same as HTTP, except that System SSL HTTPS has **Require security, KeyRing database, KeyRing password, and Use client authentication** settings.

The default port for HTTPS is 443.

HTTPS protocol settings

Select the **HTTPS** subnode to display the settings for HTTPS over SSLight. The settings are the same as HTTP, except that HTTPS has **Require security, KeyRing classname, KeyRing password, and Use client authentication** settings.

The default port for HTTPS is 443.

Trace settings

To configure the trace settings, select the **Trace** option from the **Tools** menu.

Java Gateway: Select this checkbox to enable tracing of the CICS Transaction Gateway.

This is equivalent to specifying the -trace option on the ctgstart command.

Java gateway trace file: Enter the pathname of a trace file to which trace messages will be written, if tracing is enabled. If no path is specified, the trace is written to the ctg.trc file in the bin subdirectory where CICS Transaction Gateway is installed.

You can also specify a trace file using the -tfile option on the ctgstart command.

The trace file is not appended to each time the CICS Transaction Gateway starts, the file is overwritten.

The configuration conversion tool

You use the configuration conversion tool (ctgconv) to convert the configuration files of previous versions of CICS Transaction Gateway for OS/390, to the new format of the CICS Transaction Gateway for OS/390 Version 3.1 configuration file.

The conversion tool converts the Gateway.properties file of CICS Transaction Gateway Version 3.0 and CICS Gateway for Java to an ouput file called CTG.INI by default. Samples from this file are shown in "Editing the configuration file".

The old file is renamed with the .BAK extension, but a banner is inserted into them stating that they are obsolete.

Using the conversion tool

The parameters of ctgconv are:

`ctgconv [/g=`*`filename`*`] [/o=`*`filename`*`]`

where:

/g=*filename*
Specifies the Gateway.properties file to be converted. If no . appears in the filename, it is assumed to be a directory and Gateway.properties is picked up from this directory. If the /g parameter is not specified, Gateway.properties is picked up from the current working directory.

/o=*filename*
Specifies the pathname of the converted file. If no . appears in the filename, it is assumed to be a directory and CTG.INI is written to this directory. If the /o parameter is not specified, CTG.INI is written to the current working directory.

If a converted file already exists, it is copied with the .BAK extension.

To get help on using ctgconv, enter: `ctgconv /?` or `ctgconv /h`.

Redundant parameters are removed from the old configuration files, and other parameters are given new names in the converted file.

Editing the configuration file

Although it is recommended that you use the configuration tool you can perform the configuration by editing the configuration file. A sample configuration file, ctgsamp.ini, is provided.

You must restart the CICS Transaction Gateway to pick up any changes to the configuration file.

Customizing the ctgstart script

The ctgstart script that is used to start the CICS Transaction Gateway for OS/390 is supplied in the bin directory.

You may choose to customize the EXCI options used by the CICS Transaction Gateway for OS/390 by tailoring the EXCI options table, DFHXCOPT. In this case, you must update the EXCI_OPTIONS specification in the ctgstart script to specify the library that contains the customized options table. Find the line:

```
EXCI_OPTIONS="your.user.loadlib"
```

and change it to specify the appropriate library. For information about the EXCI options and how to customize them, see the *CICS External Interfaces Guide*.

Note: If the CICS region has no security (SEC=NO), the DFHXCOPT table option SURROGCHK=NO must be specified.

You must update the high-level qualifier for the name of the library specified in EXCI_LOADLIB in the ctgstart script. This library contains:

- The default EXCI options.
- The EXCI load modules.

Find the line:

```
HLQ="CICSTS13.CICS"
```

and change it to specify the high-level qualifier of your CICS installation.

You may add export commands to the ctgstart script to set the values of environment variables for the CICS Transaction Gateway for OS/390, which are described in "Environment variables" on page 25. If you set environment variables in the ctgstart script, these override settings in the JCL.

Configuring CICS Transaction Gateway for use with RACF

The CICS Transaction Gateway for OS/390 provides support for userid and password authentication with RACF, together with the option to map a registered X.509 certificate to a RACF userid (see "Mapping client certificates to RACF userids" on page 65).

Both of these features use the OS/390 UNIX C/C++ pthread_security_np function to call RACF. This OS/390 UNIX function requires the calling address space to be marked as "program controlled". The JVM (Java Virtual Machine) that is running the CICS Transaction Gateway, invoked with ctgstart, is the calling address space in this instance.

A number of HFS files need to have an "extended attribute" set to indicate that they are program controlled. The following **extattr** commands mark the load modules used by the CICS Transaction Gateway as program controlled.

```
extattr +p /ctg/bin/lib*.so
extattr +p /ctg/bin/SECURES
```

The datasets (.SDFHEXCI) defined in your ctgstart script, referenced by the STEPLIB environment variable, also need to be marked as program controlled from an ISPF panel.

You must specify that the ctgstart script does **not** share its address space with any other processes. This is to ensure that the calling address space (JVM) is not "contaminated" by a non-program-controlled load module. To force the JVM to use its own non-sharable address space, enter:

```
extattr -s path/ctg/bin/ctgstart
```

In addition to the CICS Transaction Gateway load modules, it is also necessary to mark as program controlled certain Java HFS files that are loaded by the CICS Transaction Gateway address space. In the following example, it is assumed that a JVM is located in /usr/lpp/java118/J1.1; enter the following set of commands from this directory.

```
extattr +p bin/*
extattr +p bin/mvs/native_threads/*
extattr +p lib/mvs/native_threads/*
```

If the CICS Transaction Gateway is configured to use System SSL, binaries and datasets provided by this separate product also need to be marked as program controlled. For example:

```
extattr +p /usr/lpp/gskssl/lib/*
extattr +p /usr/lpp/gskssl/bin/*
```

Any .kdb keyring files generated using System SSL, and loaded by the CICS Transaction Gateway must also be marked as program controlled.

To use the extattr command, the user must be the owner of the file being modified, or a superuser.

Note: The userid under which the CICS Transaction Gateway is to run needs to be permitted to the BPX.SERVER FACILITY profile. For more information, see the pthread_security_np section in *OS/390 C/C++ Run-Time Library Reference*, (SC28-1663-04).

Other configuration tasks:

To complete the configuration of CICS Transaction Gateway for OS/390, you must perform the following:

- Configure CICS for the EXCI requests from the CICS Transaction Gateway. Information about this task is given in the *CICS External Interfaces Guide*.

- Create conversion templates to be used to translate the communication area used by the CICS programs requested by Web browsers. Information for this task is given in *CICS Family: Communicating from CICS on System/390*.

Chapter 5. CICS Transaction Gateway for OS/390 security

This chapter describes how to set up CICS Transaction Gateway to use the network security protocols SSL and HTTPS.

- "System SSL and SSLight" discusses and compares the different types of SSL support provided for CICS Transaction Gateway for OS/390
- "Overview" on page 40 provides an overview of network security concepts and terminology. It introduces the concepts of encryption keys, digital certificates, and KeyRings.
- "SSL and authentication" on page 44 describes the SSL protocol and the types of authentication it provides.
- "The ctgikey tool" on page 46 introduces iKeyMan, the SSLight tool provided by CICS Transaction Gateway for managing your digital certificates.
- "Using externally-signed certificates (SSLight)" on page 47 describes how to obtain externally-signed digital certificates and store them in KeyRing class files for use by the CICS Transaction Gateway.
- "Using self-signed certificates (SSLight)" on page 52 describes how to generate self-signed digital certificates for use by the CICS Transaction Gateway
- "The gskkyman tool" on page 56 introduces gskkyman, the System SSL tool provided by OS/390 V2R7.0 for managing your digital certificates.
- "Using externally-signed certificates (System SSL)" on page 57 describes how to obtain externally-signed digital certificates and store them in KeyRing database files for use by the CICS Transaction Gateway.
- "Using self-signed certificates (System SSL)" on page 61 describes how to generate self-signed digital certificates for use by the CICS Transaction Gateway
- "Mapping client certificates to RACF userids" on page 65describes the mapping of client certificates to RACF userids.
- "Configuring CICS Transaction Gateway for OS/390 for SSL and HTTPS" on page 66 describes how to configure CICS Transaction Gateway to use the secure protocols.

System SSL and SSLight

As mentioned in "Chapter 1. CICS Transaction Gateway overview" on page 1, The CICS Transaction Gateway provides two implementations of Secure Sockets Layer (SSL).

The universal implementation, written in pure Java, is referred to as **SSLight**, and is supported by a pure Java key/certificate management tool, known as iKeyman.

The second implementation, **System SSL** applies only to the CICS Transaction Gateway on OS/390 and can only be used for the SSL server. It is written in native code and supports additional cryptographic hardware accelerator cards to increase performance dramatically. It is supported by its own key/certificate management tool, known as gskkyman. (This tool is supplied with OS/390 V2R7.0.)

Therefore, for CICS Transaction Gateway for OS/390 you can configure SSL: and HTTPS: protocols handlers for System SSL or for SSLight. You should however, use System SSL in preference to SSLight for performance reasons. Configuring and using the SSLight/System SSL serverside protocol handlers are very similar, see "Configuring CICS Transaction Gateway for OS/390 for SSL and HTTPS" on page 66.

Both iKeyman and gskkyman support the use of externally signed certificates and also self-signed certificates.

In the case of the client-side code (that is, the code running the distributed applet), SSLight is always used. The strongest level of cryptographic strength currently supported by the CICS Transaction Gateway is DES 56bit. However, in the case of the CICS Transaction Gateway for OS/390 it is possible to import a VeriSign Step-Up Server certificate which will enable the Java SSLight client code (and other export-strength SSL clients) to dynamically increase their cryptographic strength (currently to 128bit).

Overview

The CICS Transaction Gateway provides comprehensive support for secure communication, which is critical to successful Internet operation. The secure network protocols allow your client applets and applications to communicate securely with your CICS Transaction Gateway using SSL. The SSL and HTTPS protocols were introduced in "Chapter 1. CICS Transaction Gateway overview" on page 1; this chapter provides more detail about setting up your CICS Transaction Gateway to use these protocols.

The following are the characteristics of secure communication:

- **Confidentiality**

  Confidentiality means that the contents of messages remain private as they pass over the Internet, or your intranet. Confidentiality is ensured through encryption of messages.

- **Integrity**

  Integrity means that messages are not altered while being transmitted. Any router along the way can insert or delete text or garble the message as it passes by. Without integrity, you have no guarantee that the message you sent matches the message received. Integrity is ensured by encryption and digital signature.

- **Accountability**

  Accountability means that both the sender and the receiver agree that the exchange took place. Without accountability, the receiver can say the message never arrived. Accountability is ensured by digital signature, so that if something goes wrong, you can identify who is accountable.

- **Authenticity**

  Authenticity means that you know who you are talking to and that you can trust that person. Authenticity requires verifying identity, so that you can make sure that others are who they say they are. Authentication is achieved by using digital signatures and digital certificates.

What is encryption?

Encryption ensures confidentiality in transmissions sent over the Internet. In its simplest form, encryption is the scrambling of a message so that it cannot be read until it is unscrambled later by the receiver. The sender uses an algorithmic pattern, or *key*, to encrypt the message, and the receiver uses a decryption key to unscramble the message.

There are two kinds of keys that can be used for encryption (as well as for digital signature and authentication):

1. Symmetric
2. Asymmetric

With *symmetric keys*, the sender and receiver share some kind of pattern, which is used by the sender to encrypt the message, and by the receiver to decrypt the message. The risk involved with symmetric keys is that you have to find a safe transportation method to use when sharing your secret key with the people with which you want to communicate.

With *asymmetric keys*, you create a key pair. This key pair consists of a *public key* and a *private key*. Unlike symmetric keys, these are different from each other, and the private key holds more of the secret encryption pattern than the public key.

A sender can broadcast its public key to whomever it wants to communicate with securely. It retains the private key and protects it with a password. Only the sender can decrypt a received message encrypted with its public key, because only it has the private key.

A protocol like SSL uses both asymmetric (also known as public key) cryptography and symmetric key cryptography. Public key cryptography is used for the TCP/IP handshake. During the handshake the master key is passed from the client to the server. The client and server make their own session keys using the master key. The session keys are then used to encrypt and decrypt data for the remainder of the session.

Digital signatures and digital certificates

A *digital signature* is a unique mathematically computed signature that ensures accountability.

A *digital certificate* allows unique identification of an entity; it is essentially an electronic ID card, issued by a trusted third party.

A digital certificate serves two purposes: it establishes the owner's identity, and it makes the owner's public key available. A digital certificate is issued by a trusted authority, a certification authority (CA), for example VeriSign Inc, Thawte. It is issued only for a limited time, and when its expiration date has passed, it must be replaced.

A digital certificate is made up of:

- The public key of the person being certified.
- The name and address of the person being certified, also known as the *Distinguished Name (DN)*.
- The digital signature of the CA.
- The issue date.
- The expiration date.

The Distinguished Name is the name and address of a person or organization. You enter your Distinguished Name as part of requesting a certificate. The digitally-signed certificate includes not only your own Distinguished Name, but the Distinguished Name of the CA, which allows verification of the CA.

To communicate securely, the receiver in a transmission must trust the CA that issued the certificate that the sender is using. As a result, any time a sender signs a message, the receiver must have the corresponding CA's *signer certificate* and public key designated as a *trusted root key*. As an example, your Web browser will have a default list of signer certificates for trusted CAs. If you want to trust certificates from another CA, you must receive a certificate from that CA and designate it as a trusted root key.

If you send your digital certificate containing your public key to someone else, what keeps that person from misusing your digital certificate and posing as you? The answer is: your private key. A digital certificate alone is never

proof of anyone's identity. The digital certificate only allows verification of the owner's identity by providing the public key needed to check the owner's digital signature. Therefore, the digital certificate owner must protect the private key that belongs with the public key in the digital certificate. Otherwise, if the private key were stolen, anyone could pose as the legitimate owner of the digital certificate.

Obtaining a digital certificate

You can obtain a certificate in two ways:

1. Buy a certificate from a CA
2. Issue yourself a certificate, that is, act as your own CA.

Buying a certificate from a CA

If you plan to conduct commercial business on the Internet, you should buy a server certificate from a CA such as VeriSign Inc (the home page is at https://www.verisign.com/).

When you submit a certificate request to VeriSign, you are expected to prove who you are before they issue you a certificate. Although the approval process is necessary to protect you, your organization, and VeriSign, it may take longer than you would like. VeriSign will digitally sign your certificate request and return the unique certificate to you through e-mail.

Note: VeriSign server certificates cannot be shared among servers on different machines.

Issuing certificates yourself

If you act as a CA, you can sign your own or anyone else's certificate request. This is a good choice if you only need the certificates within your own organization, and not for external Internet commerce. In such a scenario you might want to allow access only to a carefully controlled group of key people within your intranet.

Your key people would have browsers such as Netscape Navigator, that can receive your self-signed CA certificate and designate it as a trusted root. They would then be able to trust your communications and share information safely.

KeyRings

So where are digital certificates and their associated keys kept? The answer is that public keys, private keys, certificates and trusted root keys are kept in a *KeyRing* file.

For SSLight: The CICS Transaction Gateway uses Java classes to hold certificate and key data on both the SSL server and SSL clients. The CICS Transaction Gateway daemon (which acts as an SSL server) uses a server KeyRing, for example, **ServerKeyRing.class**, while all SSL clients use a client KeyRing, for example, **ClientKeyRing.class**. The SSL and HTTPS protocols require access to these Java classes to establish secure connections. You establish this access when you configure the CICS Transaction Gateway, see "Configuring CICS Transaction Gateway for OS/390 for SSL and HTTPS" on page 66.

For System SSL: The CICS Transaction Gateway uses a key database file to hold certificate and key data on the SSL server. Key database (.kdb) files are created using the gskkyman tool. The CICS Transaction Gateway daemon therefore, uses a server KeyRing, named for example, **ServerKeyRing.kdb**, (The .kdb file must be included in the system LIBPATH). The SSL and HTTPS protocols require access to this key database file to establish secure connections. You establish this access when you configure the CICS Transaction Gateway, see "Configuring CICS Transaction Gateway for OS/390 for SSL and HTTPS" on page 66.

Subsequent sections in this chapter tell you how to:

- Create your KeyRing files.
- Obtain your digital certificates
- Receive the digital certificates into the KeyRing files.

SSL and authentication

SSL allows the client to authenticate the identity of the server, which is called *server authentication*.

SSL Version 3 also allows the server to authenticate a client, which is called *client authentication*. This is used if the server needs to ensure who a client is before responding. If SSL client authentication is set up, the server requests the client's certificate whenever the client makes an SSL connection. The server validates the DN information in the client request with the DN information in the client's certificate before serving the document.

SSL uses a security handshake to initiate the TCP/IP connection between the client and the server. During the handshake, the client and server agree on the security keys that they will use for the session and the algorithms they will use for encryption. The client authenticates the server. In addition, if the client requests a document protected by SSL client authentication, the server

requests the client's certificate. After the handshakes, SSL is used to encrypt and decrypt all of the information in both the client request and the server response, including:

- The URL the client is requesting
- The contents of any form being submitted
- Access authorization information like user names and passwords
- All data sent between the client and the server

SSL handshaking is illustrated in Figure 8.

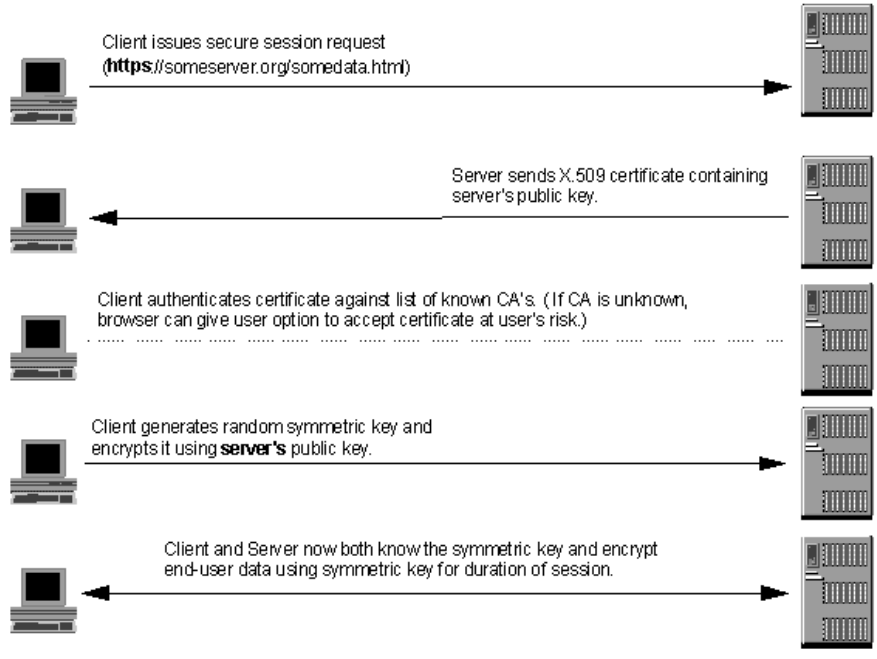

Figure 8. SSL handshake with server authentication

HTTPS

HTTPS is a unique protocol that combines SSL and HTTP. You need to specify https:// as an anchor in HTML documents that link to SSL-protected documents. A client user can also open a URL by specifying https:// to request an SSL-protected document.

Because HTTPS (HTTP + SSL) and HTTP are different protocols and usually use different ports (443 and 8080, respectively), you can run both SSL and non-SSL requests at the same time. As a result, you can choose to provide information to all users using no security, and specific information only to

browsers who make secure requests. This is how a retail company on the Internet can allow users to look through the merchandise without security, but then fill out order forms and send their credit card numbers using security.

A browser that does not have support for HTTP over SSL will naturally not be able to request URLs using HTTPS. The non-SSL browsers will not allow submission of forms that need to be submitted securely.

The ctgikey tool

CICS Transaction Gateway provides a tool, **iKeyman** for maintaining your digital certificates. You use the **ctgikey** command to set up the correct environment (including the JAVA_HOME environment variable) and invoke iKeyman.

With iKeyMan, you can:

- Request and receive a digital certificate from a CA, see "Using externally-signed certificates (SSLight)" on page 47.
- Generate self-signed certificates, see "Using self-signed certificates (SSLight)" on page 52.
- Add certificates to your KeyRing files.
- Change your KeyRing password.
- Set a private key as the default.
- Delete keys.
- Export a key by copying it to a file.
- Import a key from an exported copy and add it to a KeyRing.

Distributing iKeyman to client workstations

Although the majority of KeyRing management will be performed on the CICS Transaction Gateway machine, it may be necessary to supply the iKeyman tool to clients connecting to your SSL server. The client machines must have the minimum level of CICS Transaction Gateway Java support, which is JDK/JRE Version 1.1.8. They also require a script or command file to invoke the iKeyman Java startup class.

As an example, the following is the Windows NT command line for invoking iKeyman with the JDK:

```
java.exe -classpath
"e:\jdk118\lib\classes.zip;e:\ikeyman\cfwk.zip;e:\ikeyman\gsk4cls.jar;e:\ikeyman
\swingall.jar;" -Dkeyman.javaOnly=true com.ibm.gsk.ikeyman.Ikeyman
```

or, to invoke with the JRE:

```
jre.exe -classpath
"e:\jdk118\lib\classes.zip;e:\ikeyman\cfwk.zip;e:\ikeyman\gsk4cls.jar;e:\ikeyman
\swingall.jar;" -Dkeyman.javaOnly=true com.ibm.gsk.ikeyman.Ikeyman
```

These examples assume that the client workstation has the following files in the ikeyman directory:

- cfwk.zip
- cfwk.sec
- gsk4cls.jar
- swingall.jar
- ikminit.properties

All of this files can be found in the bin subdirectory where CICS Transaction Gateway is installed.

Note: You must ensure that cfwk.zip is the first IBM-supplied archive to appear in your CLASSPATH setting.

Using externally-signed certificates (SSLight)

The CICS Transaction Gateway can function as an SSL server, with the ability to authenticate SSL clients and accept externally-signed certificates from Certificate Authorities such as VeriSign Inc.

This section describes how to configure both an SSL server and SSL clients using the certificate management interface, iKeyMan. This is written in pure Java, so it can be distributed to a number of client/server workstations that have a suitable JVM installed.

Configuring your SSL server and clients involves: creating the KeyRing classes, obtaining the digital certificates, and receiving them into the KeyRing classes.

A **server KeyRing class** contains a Server Certificate and the corresponding private key, along with a number of signer certificates. The Server Certificate is a digital certificate that is used to identify the SSL server to connecting clients.

A **client KeyRing class** contains as a minimum, the signer certificate of the SSL server, along with a client x.509 certificate, if client authentication is required.

Configuring your SSL server

This section describes how to obtain a Trial (Test) Server Certificate from the VeriSign Web site (www.verisign.com). VeriSign allow the use of a trial Server Certificate for the period of 14 days. As this is a demonstration Server Certificate, it is not signed by the trusted VeriSign Certificate Authority, but a VeriSign Test CA.

All SSL clients will need the VeriSign Test signer certificate root key installed in their client KeyRing(s), or in the case of HTTPS connections, the repository in the browser.

The examples in this book assume you are using Netscape Communicator Version 4.5.

Obtaining a full VeriSign Server Certificate follows the same procedures as described here for the trial version.

Creating the server KeyRing

The first step is to create a server KeyRing class, which will eventually contain your signer certificates along with your Server Certificate (and its associated private-key). This repository is password-protected and you are given an indication of the password "strength" when the .class is created with iKeyMan. It is recommended that a sequence of alphanumeric characters is used; this makes the password more robust to "brute force dictionary" attacks.

To obtain the certificate:

1. Start ctgikey.
2. Select **Key Database File**.
3. From **Key Database Type**, select **SSLight key database class**.
4. Enter `ServerKeyRing.class` as the **File name**
5. In **Location**, enter a suitable location to store your ServerKeyRing.class.
6. Select **OK**.

The generated ServerKeyRing.class contains, by default, a number of the popular signer certificates including the VeriSign root signer certificate along with its Test signer certificate. It also contains VeriSign Class 1 through 4 Public Certification Authority signer certificates, which enables the Server to verify clients with VeriSign Client Certificates. This is explained in more detail in "Configuring SSL clients" on page 50.

Preparing a certificate request

Before you can obtain a Server Certificate from www.verisign.com, the SSL server must make a Certificate Request and store it locally:

1. From iKeyMan, select **Create**.
2. Select **Create New Certificate Request**.
3. You must fill out the certificate request. Some fields are optional, but you must fill in at least the following (examples are shown):

Key Label	verisignServerCert
Key Size	1024
Common Name	servermachine.hursley.ibm.com
Organization	IBM UK
Country	GB

4. Specify the name of a file in which store to store the certificate request and select **OK**. iKeyMan will then generate a public/private keypair, which may take some time depending on your processor speed.

Obtaining the Server Certificate

The next stage is to connect to the VeriSign Web site and request a Trial Server Certificate. Point your browser at http://www.verisign.com and continue to the page at: https://digitalid.verisign.com/server/trial/trialIntro.htm. You are now ready to enroll for your Trial Server Certificate.

1. Select **Continue**.
2. Select **Continue** again, and you are presented with **Step 2 of 5 - Submit CSR**.
3. Paste the contents of your certificate request file into the text box provided. The contents of your certificate request should look similar to:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIB1jCCAT8CAQAwgZUxIDAeBgNVBAMTF2NocmlzdHAuaHVyc2xleS5pYm0u
Y29tMRswGQYDVQQLExJDbGllbnRzICsgR2F0ZXdheXMxDzANBgNVBAoTBklC
TSBVSzETMBEGA1UEBxMKV0luY2hlc3RlcjEOMAwGA1UECBMFSGFudHMxETAP
BgNVBBETCFNPMjEgMkpOMQswCQYDVQQGEwJHQjCBnzANBgkqhkiG9w0BAQEF
AAOBjQAwgYkCgYEAkAth9Ar6k6ijNZ3JxdPGH6yiikwYTuA0RZDLZBSpaSEx
4qNKN/CrdF1LgfQYbZcN5NGCeC4sC478NhT+ltf5dnR3pNWBzEzmWn5mN0lH
tqJ3oibOUmDui+tQc2J9z6iRBKjkcQwjPlJp0sp5KKsev1ahAETL7LqmqMIq
pJlzKi0CAwEAAaAAMA0GCSqGSIb3DQEBBAUAA4GBAHaMHpizPs8Q3bi3I6dh
4yw0UNhojTlS1+ffiph3hK98lMHJuMztr0UMBLl/SZGNw85OJRiWuDjGYUQW
inJ0uNH34IUsnygBmt78+WlXT5nJuayg+UrAc5Ao2H8QZpRE5Sfaoc8lQcvY
plTggCdMxpYN7I33LrZDl3Po0TT8gjxQ
-----END NEW CERTIFICATE REQUEST-----
```

4. Select **Continue**.

5. The next page allows you to verify the contents of your certificate request. You must also provide the personal details requested. If a full VeriSign Server ID (rather than a trial version) were being requested these details would be used to authenticate your application.
6. When you have entered your details, and have read the VeriSign agreement, select **Accept**.
7. The next stage (**4 of 5**) is to install the Test signer certificate root on any browser that you will be using to connect to the SSL server. In the case of SSL connections from Java applets to the CICS Transaction Gateway this is not necessary, as the client applet requires a client KeyRing class, which by default will contain the Test signer certificate. However, the HTTPS protocol uses the repository in the browser, hence your need to install the Test signer certificate. Note that you do not need to install the Test signer certificate if you apply for a full VeriSign Server Certificate.

The VeriSign Trial Server Certificate will be e-mailed to the address specified in your personal details. This can typically take between one and three hours.

Receiving the Server Certificate into the server KeyRing

When you have obtained the Server Certificate, you must “receive” it into the server KeyRing using iKeyMan.

1. First copy and paste the Server Certificate data into a blank text file, using a text editor, and save the file as verisignServerID.arm.
2. From iKeyMan, select **Personal Certificates** from the pull-down menu, this is located below the **Key database content** label.
3. Select **Receive...**
4. Locate the text file containing your Server Certificate data; this will be in Base64-encoded ASCII.
5. Select **OK**.
6. Select **Key Database File** then **Exit**.

The **ServerKeyRing.class** is now ready for use with the CICS Transaction Gateway. It contains the default signer certificates along with your VeriSign Server Certificate and its corresponding private key.

Configuring SSL clients

In normal (default) operation the CICS Transaction Gateway uses only server authentication when performing an SSL handshake. and the client need only accept the presented Server Certificate. For server authentication to work, the client KeyRing class, or in the case of HTTPS, the browser certificate repository, must contain the signer certificate of the Server Certificate. In our example, the signer certificate is the VeriSign Test signer certificate.

As with the server KeyRing class, when you use iKeyMan to generate a client KeyRing class it will contain a default selection of the most popular signer certificates.

In addition to server authentication, the CICS Transaction Gateway also supports client authentication. With this option enabled, any connection attempted to either the ssl: or https: handler requires the client to present its own Client Certificate (also known as a Digital ID).

The following sections describes how to obtain a VeriSign Digital ID and generate the necessary client KeyRing class.

Obtaining the Client Certificate

In contrast to obtaining Server Certificates, it is not necessary to use iKeyMan to generate any form of certificate request. VeriSign provide a method of obtaining a Class 1 Digital ID using either Netscape or Internet Explorer browsers.

Once the Client Certificate has been obtained and installed in the browser repository, it is possible to export the certificate data, together with its associated private key, into a secure vault known as a #PKCS12 file. This file is password protected and can be imported into a client KeyRing class using the iKeyMan tool.

1. Using Netscape Communicator Version 4.5, point your browser to www.verisign.com and follow links for **Individual Certificates.**
2. Once you have reached the HTML page at https://digitalid.verisign.com/client/class1Netscape.htm, fill in the details as requested. The details will be used by VeriSign to authenticate the client application.
3. Select **Accept**.
4. Netscape will now generate a private key - it will request a password to protect this private key.
5. VeriSign will e-mail information for downloading and installing the Class 1 Digital ID to the address you submitted in your application.

During the installation process, Netscape Communicator allows you to store the Client Certificate in a password protected file (#PKCS12 format). The iKeyMan tool supports #PKCS12 format files and allows you to import certificates (along with their private key) into a client KeyRing class.

Creating the client KeyRing

To create a client KeyRing class:

1. Start ctgikey.

2. Select **Key Database File**.
3. From **Key Database Type** select **SSLight key database class**.
4. Enter `ClientKeyRing.class` as the **File name**.
5. In **Location**, enter a suitable location to store your ClientKeyRing.class.
6. Select **OK**.

A ClientKeyRing.class has now been created containing the default signer certificates, which are as follows:

```
VeriSign Class 1 Public Primary Certification Authority
VeriSign Class 2 Public Primary Certification Authority
VeriSign Class 3 Public Primary Certification Authority
VeriSign Class 4 Public Primary Certification Authority
RSA Secure Server Certification Authority
Thawte Personal Basic CA
VeriSign Test CA Root Certificate
Thawte Personal Premium CA
Thawte Premium Server CA
Thawte Personal Freemail CA
Thawte Server CA
```

Importing the Client Certificate into the client KeyRing

To import the #PKCS12 vault file containing the Client Certificate:

1. Select **Personal Certificates** from the pulldown selector below the **Key database content** label.
2. Select **Import...**
3. Set **Key file type** to `PKCS12 file`.
4. Locate the stored #PKCS12 file.
5. Select **OK**.
6. Select **Key Database File** then **Close**.
7. Select **Key Database File** then **Exit**.

The **ClientKeyRing.class** is now ready for use with the CICS Transaction Gateway. It contains the default signer certificate along with your VeriSign Client Certificate (Class 1 Digital ID) and its corresponding private-key.

Using self-signed certificates (SSLight)

The CICS Transaction Gateway provides a mechanism for you to “self-sign” your certificates. You establish yourself as your own Certification Authority and generate the X.509 digital certificates for the server (CICS Transaction Gateway) and client (browser) side.

This section describes how to configure both an SSL server and SSL clients using the certificate management interface, iKeyMan.

Configuring the SSL server

Configuring your SSL server involves: creating the server KeyRing class, generating the self-signed certificate, and receiving it into the KeyRing class.

Creating the server KeyRing

The first step is to create a server KeyRing class file to hold the Server Certificate and key information.

1. Start ctgikey.
2. Select **Key Database File - New**.
3. From **Key Database Type**, select **SSLight key database class**.
4. Enter `ServerKeyRing.class` as the **File name**.
5. In **Location**, enter a suitable location to store your ServerKeyRing.class.
6. Select **OK**.

Generating the Server Certificate

Now you are ready to create the self-signed Server Certificate and store it along with its private key in your server KeyRing class:

1. From iKeyMan, select **Personal Certificates** from the pull-down menu, this is located below the **Key database content** label.
2. Select **New Self-Signed...**
3. You must fill out the certificate details. Some fields are optional, but you must fill in at least the following (examples are shown):

Key Label	exampleServerCert
Version	X509 V3
Key Size	1024
Common Name	clientmachine.hursley.ibm.com
Organization	IBM UK
Country	GB
Validity Period	365 (days)

4. Select **OK**. iKeyMan will then generate a public/private keypair, which may take some time depending on your processor speed.
5. When iKeyman has successfully created the self-signed Server Certificate, it will appear in the Personal Certificates window. The certificate will be named according to the **Key Label** specified during the generation process, in this example `exampleServerCert.`
6. With **exampleServerCert** highlighted, select **View/Edit**. Notice that the Certificate information in the **issued to** and **issued by** textboxes are the

same, hence the certificate requester (issued to) is the same as the signer (issued by). To establish SSL connections with a server presenting this certificate, the client must trust the signer of the exampleServerCert. To do this the client key repository must contain the signer certificate of the site presenting exampleServerCert.

Exporting the Servers signer certificate

Now export the signer (public) certificate of the SSL server.

1. With **exampleServerCert** highlighted, select **Extract Certificate...**
2. Select the **Data type** of the exported certificate, this is typically Base64-encoded ASCII data.
3. Enter the name/location of the exported certificate, our example uses exampleServercert.arm.
4. Select **OK**.

The exported certificate should be stored in a safe place, and imported into any client key repository that needs to handshake with this particular SSL Server Certificate.

Configuring the SSL clients

If the SSL handler used by the CICS Transaction Gateway is configured to support only server authentication, you do not need to generate a self-signed Client Certificate. In this case the client KeyRing class need only contain the signer certificate of the Server, which is the certificate file exported in our example, exampleServercert.arm.

The following steps describe the process of creating a client KeyRing and importing the Server's signer certificate.

Creating the client KeyRing

1. Start ctgikey.
2. Select **Key Database File - New**.
3. From **Key Database Type**, select **SSLight key database class**.
4. Enter `ClientKeyRing.class` as the **File name**
5. In **Location**, enter a suitable location to store your ClientKeyRing.class.
6. Select **OK**.
7. Once the ClientKeyRing.class has been generated you should see the list of default Signers.

Importing the server's signer certificate

The next stage is to import the server's signer certificate:

1. Select **Add**.
2. Locate the stored Server Base64-encoded ASCII certificate file, in our example, `exampleServercert.arm`.
3. Give this signer certificate a unique label, for example, `My Self-Signed Server Authority`.
4. Select **OK**. This new signer certificate should be added to the list of default signers.

The generated ClientKeyRing.class can be used with the CICS Transaction Gateway's SSL protocol, which is configured to support server authentication. If the HTTPS protocol is used to establish a secure connection from the applet; the particular browser where the applet is running will need to import the exampleServercert.arm into its key/certificate repository.

Refer to your browser online help for importing Base64-encoded ASCII certificate files.

Generating a Client Certificate

Client authentication requires the client KeyRing class to also contain a self-signed Certificate that is used to identify the connecting client.

Following the same steps as for generating a self-signed Server Certificate, create (or open an existing) client KeyRing class (ClientKeyRing.class in our example), then:

1. From iKeyMan, select **Personal Certificates** from the pull-down menu, this is located below the **Key database content** label.
2. Select **New Self-Signed...**
3. Fill out the certificate details.
4. Select **OK**. iKeyMan will then generate a public/private keypair, which may take some time depending on your processor speed.
5. Like the SSL server, the client needs to install its signer certificate in the SSL server's key/certificate repository. This allows the SSL server to verify the client's details. With **exampleClientCert** highlighted, select **Extract Certificate...**
6. Select the **Data type** of the exported certificate, this is typically Base64-encoded ASCII data.
7. Enter the name/location of the exported certificate, our example uses exampleClientcert.arm
8. Select **OK**.

The exported certificate should be stored in a safe place, and imported into any client key repository that needs to handshake with this particular SSL Client Certificate.

Migrating old self-signed certificates

CICS Transaction Gateway Version 3.0 only allowed the use of self-signed certificates, and did not support externally signed certificates.

In CICS Transaction Gateway Version 3.1 you can use old self-signed certificates, because the KeyRing class files created with Version 3.0 can be imported into the iKeyman tool.

Restricting access to the server KeyRing

The contents of the server KeyRing are password encrypted; however it is highly recommended that you:

- Ensure correct file permissions are in place
- Restrict access, where applicable, to the CICS Transaction Gateway machine.

It is not good practice to share certificates among servers. You do not want servers to share a private key, particularly if they are running on different machines. A private key should never be communicated to others.

Software and hardware prerequisites for System SSL

System SSL Version 2 Release 7 is part of Cryptographic Services Base element of OS/390.

System SSL does not require any additional hardware, but performance is improved if the appropriate cryptographic hardware is installed. The Cryptographic Coprocessor Feature is an optional feature of IBM S/390 Multiprise 2000 and IBM S/390 Parallel Enterprise Server™ Generation 3 systems. It is standard on IBM S/390 Parallel Enterprise Server Generation 4 systems. The IBM 4753-014 Network Security Processor can be attached to S/390 systems that do not include the Cryptographic Coprocessor Feature.

The gskkyman tool

You can run the gskkyman tool from either an rlogin OS/390 shell environment or from the OMVS shell command-line environment.

You can use gskkyman to perform the following tasks:

The tasks that gskkyman can perform include the following:

- Creating a key database
- Creating a certificate request (and public/private key pair)
- Creating a self-signed certificate
- Exporting a certificate request to a file
- Receiving a certificate after a certificate request has been fulfilled
- Exporting a certificate to a file
- Importing a certificate from a file
- Marking a certificate as a trusted CA certificate
- Marking a certificate (and private key) as the default certificate for the key database
- Listing certificate information
- Removing a certificate (and private key) from a key database
- Removing a key database.

For further information on gskkyman, enter the following command to display help information:

```
gskkyman -h
```

or see the *Cryptographic Services System Secure Sockets Layer Programming Guide and Reference* manual, (SC24 5877).

Using externally-signed certificates (System SSL)

If you use externally-signed certificates, you must use gskkyman to set up the CICS Transaction Gateway for OS/390 as a System SSL server, as follows:

- Create a key database
- Create a certificate request
- Receive the certificate after the request has been fulfilled
- Import the certificate as a Trusted CA Certificate

Creating a key database

To create a new key database:

1. Enter gskkyman; the IBM Key Management Utility menu is displayed.
2. Enter 1 (Create new key database), then respond to prompts as follows:

```
Enter key database name or press ENTER for "key.kdb": mykey.kdb
Enter password for the key database.......> entered password
Enter password again for verification.....> entered password
Should the password expire? (1 = yes, 0 = no) [1]:
Enter password expiration time (number of days) or press ENTER
for 60 days: 35
```

3. After pressing Enter to enter the password expiration time, the following message and prompt is displayed.

```
The database has been successfully created, do you want to continue to work
with the database now? (1 = yes, 0 = no) [1]:
```

   Entering 1 or just pressing Enter, displays the key database menu, which alllows you to carry on setting up your SSL server

When you create the key database with gskkyman, you are prompted for a password that is used to protect the database. You will need to specify the password whenever you access the database. Before using the key database, you should use gskkyman to create a stashed password file, which will allow access to the database without specifying the password.

The key database is created as a file in the hierarchical file system (HFS) of OS/390 Unix System Services.

Creating a certificate request (and public/private key pair)

To create a certificate request, from the Key database menu, enter 3 (Create new key pair and certificate request), then respond to the prompts as follows:

```
Enter certificate request file name or press ENTER for "certreq.arm":
Enter a label for this key................> tjh1
Select desired key size from the following options (512):
    1:    512
    2:    1024

Enter the number corresponding to the key size you want: 1
Enter certificate subject name fields in the following.

    Common Name (required)................> servermachine.hursley.ibm.com
    Organization (required)...............> IBM
    Organization Unit (optional)..........> Hursley
    Country Name (required 2 characters)..> UK

Please wait while key pair is created...

Your request has completed successfully, exit gskkyman?

(1 = yes, 0 = no) [0]: 0
```

When creating a certificate request, you are prompted for a number of items to include in the certificate request. First you are asked for the name of a file to store the certificate request. The default name is certreq.arm. A label is

associated with every certificate and certificate request stored in the key database. You must choose a label for the certificate request. Then you are prompted for the key size and individual portions of the distinguished name that becomes part of the certificate.

Once the certificate request is created, a file with the name you specified will exist in the current working directory.

The certificate request created is stored in a file that is in base64-encoded format. This format is what is typically required by certificate authorities that create certificates. The following is the contents of the file created by the steps performed above:

```
$ cat certreq.arm
-----BEGIN NEW CERTIFICATE REQUEST-----

MIH7MIGmAgEAMEExCzAJBgNVBAYTAlVTMQwwCgYDVQQKEwNJQk0xETAPBgNVBAsT
CEVuZGljb3R0MREwDwYDVQQDEwhKb2huIERvZTBcMA0GCSqGSIb3DQEBAQUAA0sA
MEgCQQCrIZdRnXhH1EMAwTuKMKYznCFp4CFk0YG66BhvMGgfTwql9aSRWkVcer8I
I7Qk9aYzQ2LIpRhloJ9ugojy1I9VAgMBAAGgADANBgkqhkiG9w0BAQQFAANBAFc1
xOfunjyt54dUqGDdPgbnMr5A3trUhzHHkX8xlfH9Albrpsv2a3FjvnmYWFPuFXAf
3ABCD5nnsbk3AP++ic5UTM=

-----END NEW CERTIFICATE REQUEST-----
$
```

This file can either be transferred to another system (as a text file) and then transferred to the certificate authority or placed directly into a mail message sent to a certificate authority using cut-and-paste methods.

Once a certificate is created by the certificate authority in response to the request, you must received it into the key database, as described in the next section.

Receiving a certificate after a request

When the CA returns a certificate to you, you must store the certificate into the key database. Usually the CA sends an e-mail message containing a Base64-encoded certificate.

To receive the certificate, you must store the Base64-encoded certificate in an HFS file on the OS/390 system to be read in by the gskkyman command. This file should be in the current working directory when gskkyman is started.

To receive a certificate:

1. From the Key database menu, enter 4 (Receive a certificate issued for your request)
2. Respond to the promps as follows:

```
Enter certificate file name or press ENTER for "cert.arm":
Do you want to set the key as the default in your key database?
(1 = yes, 0 = no) [1]:

Please wait while certificate is received......

Your request has completed successfully, exit gskkyman?
(1 = yes, 0 = no) [0]: 0
```

You are prompted for the name of the file that contains the Base64-encoded certificate information. After receiving the certificate, you can continue to work with the key database or stop gskkyman.

Importing a certificate

If you are using a CA for generating your certificates that is not one of the default certificate authorities for which certificates are already stored in the key database, then you must import the CA's certificate into your key database file.

If you are using client authentication, then the CA certificate must be imported into the key database of the server program. The client program's key file must have the CA certificate imported regardless of whether the SSL connection uses client authentication.

A number of well-known certificate authority's (CA) certificates are stored in the key database when the key database is created. The list of CAs for which certificates are stored on key database creation is:

```
Integrion Certification Authority Root
IBM World Registry Certification Authority
Thawte Personal Premium CA
Thawte Personal Freemail CA
Thawte Personal Basic CA
Thawte Premium Server CA
Thawte Server CA
Verisign Test CA Root Certificate
RSA Secure Server Certification Authority
Verisign Class 1 Public Primary Certification Authority
Verisign Class 2 Public Primary Certification Authority
Verisign Class 3 Public Primary Certification Authority
Verisign Class 4 Public Primary Certification Authority
```

To import (store) a certificate as a CA certificate in your key database file, first get the certificate in a file in the OS/390 HFS with the file in Base64-encoded format.

To store the certificate in your key database as a CA certificate:

1. From the Key database menu, enter 6 (Store a CA certificate).
2. Respond to the propmts as follows:

```
Enter certificate file name or press ENTER for "cert.arm":
Enter a label for this key.............> xxx

Please wait while certificate is stored...

Your request has completed successfully, exit gskkyman?
(1 = yes, 0 = no) [0]: 0
```

After the operation, the certificate is treated as ″trusted″ so that it can be used in verifying incoming certificates. For a program acting as an SSL server, this certificate is used during verification of a client's certificate. For a program acting as an SSL client, this certificate is used to verify the server's certificate which is sent to the client during SSL handshake processing.

Using self-signed certificates (System SSL)

If you use self-signed certificates, you must use gskkyman to set up the CICS Transaction Gateway for OS/390 as a System SSL server, as follows:

- Create a new key database
- Create a self-signed certificate
- Export the certificate to SSL clients

Creating a key database

The procedure is the same as for externally-signed certificates.

Creating self-signed certificates

To create a certificate request, from the Key database menu, enter 5 (Create a self=signed certificate), then respond to the prompts as follows:

```
Enter version number of the certificate to be created (1, 2, or 3) [3]: 3
Enter a label for this key.................> tjh2
Select desired key size from the following options (512):
    1:    512
    2:    1024

Enter the number corresponding to the key size you want: 1
Enter certificate subject name fields in the following.

    Common Name (required).................> servermachine.hursley.ibm.com
    Organization (required)................> IBM
    Organization Unit (optional)..........> Hursley
    Country Name (required 2 characters)..> UK

Enter number of valid days for the certificate [365]:  244
Do you want to set the key as the default in your key database? (1 = yes, 0 =
no) [1]:
Do you want to save the certificate to a file? (1 = yes, 0 = no) [1]: 1
Should the certificate binary data or Base64 encoded ASCII data be saved? (1 =
ASCII, 2 = binary) [1]: 1

Enter certificate file name or press ENTER for "cert.arm":

Please wait while self-signed certificate is created...

Your request has completed successfully, exit gskkyman?

(1 = yes, 0 = no) [0]: 0
```

When creating a self-signed certificate, you are prompted for the same information requested when creating an external certificate request. In addition, you are prompted for the version number of the certificate, whether to set the certificate as the default in the key database, and whether to store the certificate in a separate file (in addition to being stored in the key database). The version number refers to the X.509 standard version number. Choose version 3. Setting the certificate as the default certificate allows the certificate to be used by the SSL Toolkit without having to specify the distinguished name in the calls to the System SSL APIs. If you choose to have the certificate saved to a file, you are prompted for some additional information regarding the file format. If you choose ASCII, a file is created whose default name is cert.arm and is found in the current working directory. The format of this file is as follows:

```
$ cat cert.arm
-----BEGIN CERTIFICATE-----
MIIBezCCASWgAwIBAgIJ+fHy9/n19/LwMA0GCSqGSIb3DQEBBQUAMEIxCzAJBgNV
BAYTAlVTMQwwCgYDVQQKEwNJQk0xETAPBgNVBAsTCEVuZGljb3R0MRIwEAYDVQQD
EwlUaW0gSm9uZXMwHhcNOTgxMjAzMTgyMjAwWhcNOTkwODA1MTgyMjAwWjBCMQsw
CQYDVQQGEwJVUzEMMAoGA1UEChMDSUJNMREwDwYDVQQLEwhFbmRpY290dDESMBAG
A1UEAxMJVGltIEpvbmVzMFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBALk1wGHpv09T
kxK1jbGfaonWM08vmJLseRm6s2ZKz81oWvOKz3eW+bsbkt8uLzS7LWEXGprvpiSi
sem6We81zxMCAwEAATANBgkqhkiG9w0BAQUFAANBACism2QymyCRZBg+oDFA8jxZ
WABC+S4kDPMkw2HcoO645PVcLstWBVol14MOZMKjLwwS291BsX1lblwWxKCI/ME=
-----END CERTIFICATE-----
$
```

If you choose binary data format, the DER-encoded certificate is stored directly to the file. The default name for this file is cert.crt and is located in the current working directory.

Exporting certificates to client programs

If you are using self-signed certificates, all SSL client programs connecting to your SSL server must be able to treat the self-signed certificate as a ″CA certificate″ This requires that the self-signed certificate be transmitted to all SSL client programs.

The first step in getting the self-signed certificate to the SSL client program's key file is to export the certificate from the SSL server's key database to a file that can be transmitted to the other program.

To export a certificate in a key database to a file:

1. From the Key database menu enter 1 (List / Manage keys and certificates). The Key and certificate list is displayed.
2. Find the label chosen when creating the certificate (tjh2 in our example) and enter the key number associated with the label.
3. Now, in the Key menu, choose 5 (Copy the certificate of this key to a file.)
4. Respond to the prompts as follows:

```
Should the certificate binary data or Base64 encoded ASCII data be saved? (1 =
ASCII, 2 = binary) [1]:1
Enter certificate file name or press ENTER for "cert.arm":

Please wait while the certificate is written to a file...

Your request has completed successfully, exit gskkyman?
(1 = yes, 0 = no) [0]: 1
```

If you choose option 1 (ASCII), for the exported certificate, the default file name is cert.arm and the resultant file is in Base64-encoded format. If you choose option 2 (binary), the default file name is cert.crt and the resultant file is in binary DER-encoded format. You can now transfer this file to the system where the SSL client program will run, and import the certificate into the SSL client program's key file as a trusted CA certificate.

Exposing client certificates

The CICS Transaction Gateway supports client authentication for both the SSL and HTTPS protocols. To enable client authentication, you must select the Use client authentication setting in the configuration tool.

When client authentication is enabled, the connecting client is requested to provide its own Client Certificate, which the server (Gateway) will validate and then expose in a user exit.

The exposure of the certificate allows the server to interrogate the certificate contents and possibly reject the connection depending upon their specified authentication rules.

System SSL

The Client Certificate is exposed by the SystemSSLServerSecurity user exit. If required, the connection is rejected by throwing an IOException within the afterDecode() method.

The server-side security class should implement the com.ibm.ctg.security.SystemSSLServerSecurity interface. The afterDecode() method exposes the GatewayRequest object, along with the com.ibm.gkssl.SSLCertificate certificate object.

The client-side security object should implement the com.ibm.ctg.security.ClientSecurity interface. For information on this interface, and the SystemSSLServerSecurity interface, refer to *CICS Transaction Gateway Programming*.

A sample, SystemSSLServerCompression.java is provided in /samples/java/com/ibm/ctg/security, and this demonstrates how the Client Certificate can be exposed.

SSLight

The Client Certificate is exposed by the SSLLightServerSecurity user exit. If required, the connection is rejected by throwing an IOException within the afterDecode() method.

The server-side security class should implement the com.ibm.ctg.security.SSLightServerSecurity interface. The afterDecode() method exposes the GatewayRequest object, along with the com.ibm.sslight.SSLCert[] certificate chain object.

The client-side security object should implement the com.ibm.ctg.security.ClientSecurity interface. For information on this interface, and the SSLightServerSecurity interface, refer to *CICS Transaction Gateway Programming*.

A sample, SSLightServerCompression.java is provided in /samples/java/com/ibm/ctg/security, and this demonstrates how the Client Certificate can be exposed.

Mapping client certificates to RACF userids

You can associate a client certificate with a RACF userid by running the RACDCERT command under TSO. (This command does not run under the OpenEdition shell.)

You should already have performed the actions described in "Configuring CICS Transaction Gateway for use with RACF" on page 36.

Before executing RACDCERT, you must download (with RECFM=VB) the certificate that you wish to process into an MVS sequential file that is accessible from TSO. The syntax of RACDCERT is:

```
RACDCERT ADD('datasetname') TRUST [ ID(userid) ]
```

where *datasetname* is the name of the dataset containing the client certificate, and *userid* is the userid that is to be associated with the certificate. If the optional ID(*userid*) parameter is omitted, the certificate is associated with the user issuing the RACDCERT command.

After you issue the RACDCERT command, RACF creates a profile in the DIGTCERT class that makes the association between certificate and userid. This profile can then be used to translate a certificate to a userid without a password being supplied.

For further information on the RACDCERT command, including the format of data allowed in the downloaded certificate dataset, see the *OS/390 Security Server (RACF) Command Language Reference* book.

CICS Transaction Gateway provides the class com.ibm.ctg.util.RACFUserid for mapping an X.509 certificate to a RACF userid. The class has the following methods:

- **setCertificate(byte[] clientCertificateData);**
- **getRACFUserid();**

The user can create a RACFUserid object in two ways:

1. `RACFUserid myUseridObject = new RACFUserid();`
2. `RACFUserid myUseridObject = new RACFUserid(myCertificate);`

The second approach is preferred because it will construct an object and automatically populate it with certificate data, without needing to call the **setCertificate(byte[] clientCertificateData);** method. When the object is created, **getRACFUserid()** makes a native OS/390 call and attempts to map the certificate supplied to a RACF userid. (assuming the user has already established a link between an X.509 certificate and a valid RACF userid).

The **getRACFUserid()** method will return a string containing the RACF userid.

The SystemSSLServerCompression.java class in the samples/java/com/ibm/ctg/security directory gives an example of creating the RACFUserid object.

For more information on the com.ibm.ctg.util.RACFUserid class, refer to the HTML programming reference information. For more information on programming samples refer to the *CICS Transaction Gateway Programming* book.

Configuring CICS Transaction Gateway for OS/390 for SSL and HTTPS

To use the SSL and HTTPS protocols, you must enable them using the configuration tool (see "Using the configuration tool" on page 27). You can enable the following:

SSL SSLight.

System SSL
System SSL.

HTTPS
HTTP over SSLight.

System HTTPS
HTTP over System SSL.

This creates the correct handler definitions in the CTG.INI file.

The protocols will then be started when the CICS Transaction Gateway is executed. You can specify that either SSL or HTTPS is used, or both, and it is possible to use SSLight and System SSL at the same time. When the CICS Transaction Gateway is started, it listens for SSL requests on port 8050, and for HTTPS requests on port 443, as long as the handlers are enabled.

Howevever, if you use SSLight and System SSL handlers concurrently, make sure that unique port numbers are specified.

The settings that are specific to the SSL and HTTPS protocols are:

KeyRing database
This setting specifies the name of the System SSL server key database.

KeyRing classname
For SSLight, this setting specifies the name of the SSL server KeyRing classfile. CLASSPATH must be set so that this class can be found.

KeyRing password
This setting specifies the password used to decrypt the encrypted server KeyRing.

Use client authentication
This setting specifies that client authentication is to be used. The default is that server authentication is used.

Using the configuration tool will create the necessary entries in the configuration file. Full examples of SSL and HTTPS protocol entries are contained in the sample configuration file CTGSAMP.INI.

The following is an example of a protocol handler entry in CTG.INI for SSLight with client authentication enabled:

```
protocol@ssl.handler=com.ibm.ctg.server.SslHandler
protocol@ssl.parameters=port=8050;sotimeout=1000;connecttimeout=2000;
idletimeout=600000;pingfrequency=60000;keyring=ServerKeyRing;keyringpw=default;
clientauth=on;
```

and the following is an example for System SSL:

```
protocol@systemssl_ssl.handler=com.ibm.ctg.server.GskSslHandler
protocol@systemssl_ssl.parameters=port=8050;sotimeout=1000;connecttimeout=2000;
idletimeout=600000;pingfrequency=60000;keyring=key.kdb;keyringpw=password;
clientauth=on;
```

For SSLight, the CICS Transaction Gateway provides two default KeyRing class files that can be used to establish SSL and HTTPS connections. The **ClientKeyRing** and **ServerKeyRing** are both encrypted using the password **default**, and are only recommended for use in testing environments. Therefore, to use the SSL and HTTPS protocols, we recommend that you generate your own KeyRings, as described in preceding sections.

Specifying the client KeyRing

Which secure protocol is used will determine whether a client KeyRing is required. The HTTPS protocol is designed for secure communication from within a Java applet, where the browser (client) itself has the necessary functionality to establish a secure connection with the CICS Transaction Gateway (server). For this reason the HTTPS protocol only requires a server-side KeyRing to be specified, the client side is handled by the browser software.

The SSL protocol is designed at a much lower level in which the CICS Transaction Gateway has code to handle the server and the client in a secure fashion. The SSL protocol requires a KeyRing for both the server *and* the client.

The client KeyRing is specified by setting a static field in the SslJavaGateway.class. This class forms part of the CICS Transaction Gateway client-side code.

The SslJavaGateway.class provides two methods: one for ”getting” and one for “setting” the client KeyRing:

```
public static void setKeyRing(String strSetKeyRing, String strSetKeyRingPW)
public static String getKeyRing()
```

To set the client KeyRing class to be used by the SSL protocol, your client application or applet would make a static call to the following method:

SslJavaGateway.setKeyRing(CLASSname, PASSword);

where:

- CLASSname denotes the classname of the Java KeyRing class generated for the client
- PASSword is used to decipher the embedded X.509 certificate.

The SslJavaGateway.class also provides the “getter” method **getKeyRing()** to return the CLASSname of the currently specified client KeyRing.

Using the SSL/HTTPS protocols to establish a connection from a client application or applet to the CICS Transaction Gateway is no different from using the TCP or HTTP protocols. The client application or applet simply “flows” its request to the CICS Transaction Gateway using the relevant URL. For example, for SSL the application would use `ssl://transGatewayMachine:8050`, or for HTTPS it would use `https://transGatewayMachine:443`.

See the *CICS Transaction Gateway Programming* book and the CICS Transaction Gateway programming interface HTML pages for further information regarding the design and implementation of client-side programs.

Chapter 6. CICS Transaction Gateway for OS/390 operation

This chapter describes how to start and stop the CICS Transaction Gateway for OS/390. The startup options that you can specify are described.

Starting the CICS Transaction Gateway for OS/390

You can start the gateway:
- From an OS/390 UNIX® System Services command line
- By submitting JCL to start a batch job.

You can start the gateway with default values for the options, or with user-supplied values. The options and their values are described in "Starting the Gateway with user-specified options".

Starting from a command line

To start the CICS Transaction Gateway for OS/390 with the default options, type `ctgstart` at the command prompt and press Enter. You see the startup message:

```
CCL6500I: Starting the Gateway with default values.
```

This is followed by two lines showing the values that are being used, for example:

```
CCL6502I: [ Initial ConnectionManagers = 1, Maximum ConnectionManagers = 100,
CCL6502I: Initial Workers = 1, Maximum Workers = 100, Port = 2006 ]
```

Starting the Gateway with user-specified options

The user definable options on the start command are:

-port=*port_number*
The TCP/IP port number assigned to the CICS Transaction Gateway for OS/390

-initconnect=*number*
The initial number of ConnectionManager threads.

-maxconnect=*number*
The maximum number of ConnectionManager threads. If this value is set to -1, no limits are applied to the number of ConnectionManager threads.

-initworker=*number*
The initial number of Worker threads.

-maxworker=*number*
The maximum number of Worker threads. If this value is set to -1, no limits are applied to the number of Worker threads.

-trace
Enables extra tracing messages.

-notime
Disables timing information in messages (times are shown to millisecond accuracy).

-tfile=*pathname*
If tracing is enabled, writes the trace messages to the file specified in *pathname*. If no path is specified, the trace is written to the ctg.trc file in the bin subdirectory where CICS Transaction Gateway is installed.

-noinput
Disables the reading of input from the console.

-nonames
Enables or disables the lookup of TCP/IP host names for printing the connect and disconnect messages. This can improve performance, and it allows the CICS Transaction Gateway for OS/390 to be used on systems that have no domain name server.

-x Enables full debug tracing. This includes everything traced by the -trace option, plus additional information. This option will decrease performance significantly.

To override the startup defaults, type: `ctgstart` at the command prompt, followed by the startup options you require, and press Enter.

You see the startup message:

```
CCL6501I: Starting the CICS Transaction Gateway with user specified values.
```

This is followed by two lines showing the values that are being used, for example:

```
CCL6502I: [ Initial ConnectionManagers = 10, Maximum ConnectionManagers = 100,
CCL6502I: Initial Workers = 10, Maximum Workers = 100, Port = 2006 ]
```

To get help for the startup options, enter: `ctgstart ?`

Starting with JCL

This is a sample of the JCL you can use to start the CICS Transaction Gateway for OS/390 as a batch job:

```
//DFHJGATE JOB (Accounting info),CLASS=A,USER=user,PASSWORD=passwd,
//         MSGCLASS=H
//OEEXCI   EXEC PGM=BPXBATCH,
//           PARM='SH cd /ctg/bin;ctgstart -noinput',
```

```
//           REGION=8M
//STDIN  DD PATH='/dev/null',
//         PATHOPTS=(ORDONLY)
//STDOUT DD PATH='/jgateo.log',PATHOPTS=(OWRONLY,OCREAT),
//           PATHMODE=SIRWXU
//STDERR DD PATH='/jgatee.log',PATHOPTS=(OWRONLY,OCREAT),
//           PATHMODE=SIRWXU
//STDENV DD *
DFHJVPIPE=JAVAGAT1
DFHJVSYSTEM_00=IJKLMNOP-Primary CICS server
DFHJVSYSTEM_01=OTHER-Some other CICS system
/*
//
```

In this example:

- BPXBATCH is the MVS program that runs an OS/390 UNIX® System Services script as a batch job. The PARM field specifies that the shell (SH) is to execute the specified `ctgstart` command with the `-noinput` option. The path assumes that the top-level /ctg directory is directly accessible from the HFS root.
- STDENV is a data set in which you can set the values of any environment variables. In the example, the DFHJVPIPE, DFHJVSYSTEM_00, and DFHJVSYSTEM_01 environment variables are set (see “Environment variables” on page 25 for more information). The values to be given to the environment variables must not be enclosed in quotes. If a line has a sequence numbers, the sequence number is interpreted as part of the value of the environment variable. Values specified in JCL are overridden by values specified in the ctgstart script (see “Customizing the ctgstart script” on page 35).

Stopping the CICS Transaction Gateway for OS/390

If you started the CICS Transaction Gateway for OS/390 from the command line, and if you did not specify the `-noinput` parameter, you can stop the Gateway by typing `Q` and pressing Enter in the Gateway console session.

If you have used the `-noinput` parameter, or if you used JCL to start the CICS Transaction Gateway for OS/390, you must stop the Gateway process by using the JES CANCEL command.

Chapter 7. CICS Transaction Gateway for OS/390 programming overview

This chapter provides an introduction to Java programming with the CICS Transaction Gateway for OS/390. Detailed information on programming is given in the *CICS Transaction Gateway Programming* book, although not all of the information in that book is relevant to the OS/390 platform.

Programming interface

The CICS Transaction Gateway for OS/390 provides the following classes and interfaces, which make up its public programming interface.

- **Java client program classes**
  - com.ibm.ctg.client.JavaGateway
  - com.ibm.ctg.client.ECIRequest
  - com.ibm.ctg.client.EPIRequest
  - com.ibm.ctg.client.ESIRequest
  - com.ibm.ctg.client.CicsCpRequest
  - com.ibm.ctg.client.Callbackable (interface)
  - com.ibm.ctg.client.GatewayRequest
- **Interface definitions and certificate objects for writing Gateway security classes**
  - com.ibm.ctg.security.ClientSecurity
  - com.ibm.ctg.security.ServerSecurity
  - com.ibm.ctg.security.SSLightServerSecurity
  - com.ibm.sslight.SSLCert[]
  - com.ibm.ctg.security.SystemSSLServerSecurity
  - com.ibm.gskssl.SSLCertificate
  - com.ibm.ctg.util.RACF.Userid

Writing Java client programs

At the simplest level, the flow of program control needed to write a simple CICS Transaction Gateway for OS/390 Java client program is as follows:

1. The Java program creates and opens an instance of a com.ibm.ctg.client.JavaGateway object.
   - The default JavaGateway constructor creates a blank JavaGateway object. You **must** then set the correct properties in this object using the relevant set.. methods. The JavaGateway is then opened by calling the open method.

- Two other JavaGateway constructors exist that simplify the creation of a JavaGateway by setting the relevant properties and implicitly calling the open method for you. On return from a successful call to one of these constructors, the resultant JavaGateway is open and connected to the requested CICS Transaction Gateway for OS/390.

2. The Java program creates an instance of the com.ibm.ctg.client.ECIRequest object containing the request that it wishes to make.
3. The Java program then flows the request to the CICS Transaction Gateway for OS/390 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 program continues to create request objects and flow them through the JavaGateway object, as appropriate.
6. The Java program then closes the JavaGateway object.

The CICS Transaction Gateway for OS/390 also provides a sample program TestECI, to illustrate the use of these classes.

TestECI

TestECI is a sample program that allows you to test the functionality of the CICS Transaction Gateway for OS/390. With TestECI you can connect to a Gateway and then send one or more ECI requests to a CICS server. If you specify more than one CICS program on the server, all the programs are run as one extended Logical Unit of Work (LUW). You can run TestECI either as an application, or as an applet.

The source for TestECI is provided in the directory:

```
/ctg/samples/java/com/ibm/ctg/test
```

Running TestECI as an application

When running TestECI as an application, parameters are passed in via the command line and output appears in the console.

The syntax is:

```
java com.ibm.ctg.test.TestECI [jgate=jgate_URL]
                              [jgateport=jgate_port]
                              [clientsecurity=client_security_class]
                              [serversecurity=server_security_class]
                              [server=cics_server]
                              [userid=cics_userid]
                              [password=cics_password]
                              [prog<0..9>=prog_name]
```

```
[commarea=comm_area]
[commarealength=comm_area_length]
[status]
[trace]
```

Where :

- *jgate_URL* is the URL of the Gateway to connect to.
- *jgate_port* is the TCP/IP port to connect to on jgate_server, if it was not specified as part of the *jgate_URL*.
- *client_security_class* is the name of the class to use to provide client-side security
- *server_security_class* is the name of the class to use to provide server-side security
- *cics_server* is the name of the CICS server to receive ECI requests.
- *cics_userid* 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.
- *comm_area* is the initial value of the COMMAREA, if any
- *comm_area_length* is the length of the COMMAREA to send to each CICS server program
- `status` causes the program to query the status of all the known CICS servers.
- `trace` causes tracing information to be produced.

For example:

```
java com.ibm.ctg.test.TestECI jgate=myjgate.here.com server=mycics
commarea="Hello World" prog0=testprog prog1=testprog2 status
```

Running TestECI as an applet

When running TestECI as an applet you pass in parameters via <param> tags within the <applet> tag. Output appears in a text area on the browser running the applet.

The parameters are the same as those used when running TestECI as an application (see “Running TestECI as an application” on page 76).

```
<applet code="TestECI" align="baseline" width="128 height="128" .....>

<param name="jgate" value="jgate_URL">
<param name="jgateport" value="jgate_port">
<param name="clientsecurity" value="client_security_class">
<param name="serversecurity" value="server_security_class">
<param name="server" value="cics_server">
<param name="userid" value="cics_userid">
<param name="password" value="cics_password">
<param name="progn" value=prog_name">
```

```
<param name="commarea" value="comm_area">
<param name="commarealength" value="comm_area_length">
<param name="status" value="yes">
<param name="trace" value="yes">
</applet>
```

If *jgate_URL* is not specified, the browser connects to the applet host.

Sample HTML to invoke TestECI as an applet is provided in testeci.html, which is located with the TestECI source.

Making ECI calls

The CICS Java classes allow you to create **ECIRequest** objects that represent calls to the ECI.

Program link calls

You can create **ECIRequest** objects that represent requests for synchronous and asynchronous program link calls (ECI_SYNC and ECI_ASYNC call types).

The following restrictions apply to the CICS Transaction Gateway for OS/390:

- The length of the communication area (COMMAREA) passed to the ECI must not exceed 32 659 bytes.
- You must specify a callback routing with an asynchronous call. Reply solicitation calls are not supported.
- You can specify any value for **eci_message_qualifier**, but it is ignored.

Status information calls

The ECI status information calls (ECI_STATE_SYNC and ECI_STATE_ASYNC call types) return their results in a CICS communication area. Since the contents of the communication area are platform dependent, the **ECIRequest** class provides help with interpreting the results of a status information call:

- Public variables to hold the results of a status information call in a platform-independent manner
- Two call types (ECI_STATE_SYNC_JAVA and ECI_STATE_ASYNC_JAVA) that return the results of a status information call in the public variables rather than in a communication area
- Methods to interpret the contents of the public variables as strings for display

The CICS Transaction Gateway for OS/390 supports only the ECI_STATE_IMMEDIATE value for the extend mode in status information calls.

Reply solicitation calls

You can create **ECIRequest** objects that represent reply solicitation calls.

The CICS Transaction Gateway for OS/390 does not support reply solicitation calls.

CICS security considerations

The CICS Transaction Gateway for OS/390 region is an EXCI user, so the following security considerations apply:

- If the user ID of the CICS Transaction Gateway for OS/390 address space is *the same as* the user ID of the CICS address space:
  - LINK security checking is not performed.
  - Surrogate user checking is performed, so the user ID of the CICS Transaction Gateway for OS/390 address space must be authorized to use the user ID passed on the ECI call.
- If the user ID of the CICS Transaction Gateway for OS/390 address space is *different from* the user ID of the CICS address space, LINK security checking is performed according to the setting of ATTACHSEC for the connection. The user ID passed on the ECI call is validated.
- The userid and password coded on the ECIRequest object is validated with RACF for each and every EXCI call.
- Userid and password authentication can be disabled for each EXCI call. This is done using the AUTH_USERID_PASSWORD environment variable in the ctgstart script. See “Environment variables” on page 25 for more information.

ECI return codes and server errors

This section describes how the return codes from the EXCI are returned to the user of the **ECIRequest** object.

Table 2 shows how EXCI return codes map to ECI return codes. The EXCI return codes are documented in the *CICS External Interfaces Guide*.

Table 2. EXCI return codes and ECI return codes

EXCI return codes	ECI symbolic names/return codes	rc
201, 203	ECI_ERR_NO_CICS	—3
202	ECI_ERR_RESOURCE_SHORTAGE	—16
401, 402, 403, 404, 410, 411, 412, 413, 418, 419, 421	ECI_ERR_SYSTEM_ERROR	—9

Table 2. EXCI return codes and ECI return codes (continued)

EXCI return codes	ECI symbolic names/return codes	rc
422	ECI_ERR_TRANSACTION_ABEND	—7
423	ECI_ERR_SECURITY_ERROR	—27
601, 602, 603, 604, 605, 606, 607, 608, 621, 622, 623, 627, 628	ECI_ERR_SYSTEM_ERROR	—9
609	ECI_ERR_SECURITY_ERROR	—27
624	ECI_ERR_REQUEST_TIMEOUT	—5

Making EPI calls

The CICS Java classes allow you to create **EPIRequest** objects that represent calls to the EPI. You set the public variable `Call_Type` in an **EPIRequest** object to specify which EPI call you wish to make. The results of the EPI call are returned in the object after you use the **flow** method of the **JavaGateway** object.

However, as you might expect, the CICS Transaction Gateway for OS/390 rejects attempts to flow an **EPIRequest** object. The return code EPI_ERR_FAILED is returned. You should not use **EPI_GetSysError** to attempt to get more information. If you wish to run transactions in the manner of the EPI, you should use the ECI and set up a request for DFHWBTTA. This is described in the *CICS Internet Guide*.

Chapter 8. CICS Transaction Gateway for OS/390 problem determination

Problem determination is not to be confused with problem solving, although while investigating a problem you may find enough information to solve the problem. Examples of the types of problem that can arise are:

- End-user errors
- Programming errors
- Configuration errors.

Preliminary checks

Before investigating the problem, it is worth checking to see whether there is an obvious cause:

- Has the system run successfully before?
- Have you made any changes to the configuration of the system or added new features or programs?
- If you have migrated from using the old CICS Gateway for Java (MVS) or CICS Transaction Gateway for OS/390 Version 3.0, are you sure that the configuration file (Gateway.properties), and JGate script were migrated successfully?
- Are your environment variables, such as CLASSPATH and LIBPATH set correctly? (See “Environment variables” on page 25.)
- Are there any messages explaining the failure?
- Can the failure be reproduced?

You should also check the log files ./ctg/jgateo.log and ./ctg/jgatee.log.

Problems running sample applets using the JDK AppletViewer

When using AppletViewer to run any of the CICS Transaction Gateway sample applets from your local filesystem, the message CCL6664E `Unable to load relevant class to support the` *xyz* `protocol` may be displayed.

To resolve this problem, select **Applet** from the AppletViewer menu bar; then select **Properties**. Then set both **Network Access** and **Class Access** to `Unrestricted`.

Conflicts with default ports

The CICS Transaction Gateway uses default ports for its supported protocols. These may conflict with ports already in use, and if this is the case, one or more protocols fail to start successfully. You can change port number in the CTG.INI file, see "Using the configuration tool" on page 27.

What to do next

If you think the problem is in the CICS Transaction Gateway for OS/390, you need to collect as much information as possible and contact your support organization.

If you started the Gateway without **trace** enabled, you need to stop the Gateway, restart it with the **trace** option, and recreate the problem.

If you suspect the problem is elsewhere in the network, you should follow the problem determination procedures provided with those other products. See the *CICS Problem Determination Guide*.

Program support

Different levels of program support are available and you should check what level you have before contacting IBM. Warranty and support information is provided in the *License* documents you get with the product; some products also include a **Service and Support** card, or visit our Web site at:

`http://www.ibm.com/software/ts/cics/`

and follow the **Support** link.

Messages

The CICS Transaction Gateway for OS/390 messages have a *CCL* prefix; which has traditionally been used for CICS Clients.

For a list of the messages generated by the CICS Transaction Gateway for OS/390, see the *CICS Transaction Gateway Messages* book.

Sources of information

You can get problem determination information from the following sources:

- The standard output (stdout) and standard error (stderr) files:
  - Standard output contains a log of CICS Transaction Gateway messages, and any messages returned to the EXCI from CICS.
  - Standard error contains the messages in standard output, together with error messages from the Java virtual machine. The Java virtual machine messages are documented in the Java development toolkit (JDK).
- CICS Transaction Gateway for OS/390 trace: This shows the activity in the CICS Transaction Gateway for OS/390 when it is handling a request from a Web browser. This trace is written as to the EXCI trace, and the trace points are listed in Table 3 on page 84.
- EXCI trace: This shows the EXCI activity. The CICS Transaction Gateway for OS/390 trace is part of the EXCI trace. For details of EXCI tracing, see the *CICS External Interfaces Guide*
- CICS trace: This shows the progress of the request through:
  - The CICS mirror transaction that handles the EXCI request
  - The called CICS program.

  You should examine the AP trace points in the CICS trace.

To route messages and trace information to a file, specify:

```
ctgstart -trace -tfile
```

when starting the Gateway, which generates a ctg.trc file in the bin directory.

To produce a full debug trace, you can use the `ctgstart -x` option.

Tracing in the CICS Transaction Gateway for OS/390

The CICS Transaction Gateway for OS/390 writes trace entries to a buffer in its address space. The trace entries are in the CICS trace EXCI format, so the trace entries in a dump can be printed using standard OS/390 utilities. You can use the following operating procedure from the SDSF operator console:

1. Use the command

   ```
   /D OMVS,A=ALL
   ```

   to display OMVS tasks.
2. Find the CICS Transaction Gateway for OS/390 task, and note the ASID.
3. Enter the DUMP command with a suitable comment. For example:

   ```
   /DUMP COMM=(JGATE DUMP)
   ```

4. Reply to the message with the ASID as follows:

   `/R nn,ASID=aa,END`

   where *nn* is the message number for the reply, and *aa* is the ASID.

Table 3 shows the trace points written to the EXCI trace by the CICS Transaction Gateway for OS/390.

Table 3. EXCI trace points for the CICS Transaction Gateway for OS/390

Point ID	Module	Lvl	Type	Data
8000	JVDLL	EX 1	ECI parameters passed	**1** Thread name **2** `Call_Type` **3** `Extend_Mode` **4** `Luw_Token` **5** `Commarea_Length` **6** `Cics_Rc` **7** `Message_Qualifier`
8001	JVDLL	Exc	GetStringPlatform error	**1** Return code **2** Data area **3** Length
8002	JVDLL	Exc	GetStringPlatformLength error	**1** Return code
8003	JVDLL	EX 1	Converted parameter	**1** Thread name **2** Parameter name **3** Parameter value
8004	JVDLL	EX 1	Inbound COMMAREA	**1** Thread name **2** Communication area length **3** Communication area address **4** 250 bytes of data
8006	JVDLL	EX 1	Outbound COMMAREA	**1** Thread name **2** Communication area length **3** Communication area address **4** 250 bytes of data

Table 3. EXCI trace points for the CICS Transaction Gateway for OS/390 (continued)

Point ID	Module	Lvl	Type	Data
8007	JVDLL	EX 1	ECI parameters output	**1** Thread name **2** Call_Type **3** Extend_Mode **4** Luw_Token **5** Commarea_Length **6** Cics_Rc **7** Message_Qualifier
8010	JVDLL	Exc	Error response received	**1** Function number **2** EXCI response **3** EXCI reason **4** EXCI subreason field-1 **5** EXCI subreason field-2 **6** Cics_Rc
8011	JVDLL	Exc	DPL_REQUEST error	**1** RESP **2** RESP2 **3** ABCODE **4** Cics_Rc
8012	JVDLL	Exc	WBA1 parameters allocation error	**1** malloc length **2** Cics_Rc
8013	JVDLL	Exc	Invalid call type	**1** Thread name **2** Call_Type **3** Cics_Rc
8014	JVDLL	Exc	Invalid COMMAREA length	**1** Thread name **2** Commarea_Length **3** Size of Commarea **4** Cics_Rc

Appendix A. The CICS Transaction Gateway and CICS Universal Clients library

This chapter lists all the CICS Transaction Gateway, CICS Universal Clients, and related books, and discusses the various forms in which they are available.

The headings in this chapter are:

CICS Transaction Gateway books

- *CICS Transaction Gateway for OS/2 Administration, SC34-5590*

  This book describes the administration of the CICS Transaction Gateway for OS/2.

- *CICS Transaction Gateway for Windows Administration, SC34-5589*

  This book describes the administration of CICS Transaction Gateway for Windows 98 and CICS Transaction Gateway for Windows NT.

- *CICS Transaction Gateway for AIX Administration, SC34-5591*

  This book describes the administration of the CICS Transaction Gateway for AIX.

- *CICS Transaction Gateway for Solaris Administration, SC34-5592*

  This book describes the administration of the CICS Transaction Gateway for Solaris.

- *CICS Transaction Gateway for OS/390 Administration*, SC34-5528 *CICS Transaction Gateway for OS/390 Administration, SC34-5528*

  This book describes the administration of the CICS Transaction Gateway for OS/390.

- *CICS Transaction Gateway Messages*

  This online book lists and explains the error messages that can be generated by CICS Transaction Gateway.

  You cannot order this book.

- *CICS Transaction Gateway Programming*, SC34-5594

  This book provides an introduction to Java programming with the CICS Transaction Gateway.

  There are also additional HTML pages that contain programming reference information.

CICS Universal Clients books

- *CICS Universal Client for OS/2 Administration, SC34-5450*

  This book describes the administration of the CICS Universal Client for OS/2.

- *CICS Universal Client for Windows Administration, SC34-5449*

  This book describes the administration of the CICS Universal Client for Windows 98 and CICS Universal Client for Windows NT.

- *CICS Universal Client for AIX Administration, SC34-5348*

  This book describes the administration of the CICS Universal Client for AIX.

- *CICS Universal Client for Solaris Administration, SC34-5451*

  This book describes the administration of the CICS Universal Client for Solaris.

- *CICS Universal Clients Messages*

  This online book lists and explains the error and trace messages that can be generated by CICS Universal Clients.

  You cannot order this book.

- *CICS Universal Clients C++ Programming*, SC33-1923

  This book describes how to write object oriented programs for the ECI and EPI in the C++ language.

- *CICS Universal Clients COM Automation Programming*, SC33-1924

  This book describes how to write object oriented programs for the ECI and EPI according to the Component Object Model (COM) standard.

CICS Family publications

- *CICS Family: Client/Server Programming*, SC33-1435

  This book describes the programming interfaces associated with CICS client/server Programming— the External Call Interface (ECI), the External Presentation Interface (EPI), and the External Security Interface (ESI). It is intended for application designers and programmers who wish to develop client applications to communicate with CICS server systems.

Book filenames

Table 4 show the softcopy filenames of the CICS Transaction Gateway and CICS Universal Client books.

Table 4. CICS Transaction Gateway and CICS Universal Clients books and file names

Book title	File name
CICS Universal Clients Messages	CCLHAB
CICS Universal Client for AIX Administration	CCLHAD
CICS Universal Client for OS/2 Administration	CCLHAE
CICS Universal Client for Windows Administration	CCLHAF
CICS Universal Client for Solaris Administration	CCLHAG
CICS Transaction Gateway for OS/390 Administration	CCLHAI
CICS Transaction Gateway Messages	CCLHAJ
CICS Transaction Gateway Programming	CCLHAK
CICS Transaction Gateway for Windows Administration	CCLHAL
CICS Transaction Gateway for OS/2 Administration	CCLHAM
CICS Transaction Gateway for AIX Administration	CCLHAN
CICS Transaction Gateway for Solaris Administration	CCLHAO
CICS Universal Clients C++ Programming	CCLHAP
CICS Universal Clients COM Automation Programming	CCLHAQ
CICS Family: Client/Server Programming	DFHZAD
Note: The File names in this table do not include the 2-digit suffix.	

Sample configuration documents

A number of sample configuration documents are available in the Portable Document Format (PDF) format.

These documents provide step-by-step guidance to help you, for example, in configuring your CICS Universal Clients for communication with CICS servers, using various protocols. They provide detailed instructions that extend the information in the CICS Transaction Gateway and CICS Universal Client libraries.

As more sample configuration documents become available, you can download them from our Web site; go to:

`http://www.ibm.com/software/ts/cics/`

and follow the **Library** link.

Other publications

The following International Technical Support Organization (ITSO) Redbook publication contains many examples of client/server configurations:

- *Revealed! CICS Transaction Gateway with more CICS Clients Unmasked, SG24-5277*

  This book supersedes the following book:

- *CICS Clients Unmasked, GG24-2534*

You can obtain ITSO Redbooks from a number of sources. For the latest information, see:

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

You can find information on CICS products at:

`http://www.ibm.com/software/ts/cics/`

Viewing the online documentation

You can access all of the documentation provided with CICS Transaction Gateway and CICS Universal Client in our online library. You need Adobe Acrobat Reader and a suitable Web browser to use the online library (and you may need to configure these).

To get to the online library:

- On Windows and OS/2, select the **Documentation** icon.
- On AIX and Solaris, run the **ctgdoc** script.

and the library home page is displayed.

The online library allows you to link to:

- CICS Transaction Gateway and CICS Universal Clients books in PDF format.
- Programming reference documentation in HyperText Markup Language (HTML) files (provided for CICS Transaction Gateway only).
- README files.
- Sample configuration documents in PDF format.
- Translated books in PDF format. (You may find that not all books are translated for your language.)
- The CICS Web site.

Guidance information on using Acrobat Reader is also provided.

Updated versions of the books may be provided from time to time, check our Web site at:

`http://www.ibm.com/software/ts/cics/`

and follow the **Library** link.

Viewing PDF books

The PDF information provides powerful functions for:

- Navigating through the information. There are hypertext links within PDF documents, and to other PDF documents and Web pages.
- Searching for specific information.
- Printing all or part of PDF documents on a PostScript printer.

You can find out more about Acrobat Reader at the Adobe Web site:

`http://www.adobe.com/acrobat/`

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply in the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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 United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX	CICS
IBM	MVS
OpenEdition	OS/2
OS/390	System/390
TXseries	WebSphere

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, or other countries, or both.

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.

Index

T

U

V

W

Program Number: 5648-B43

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC34-5528-01