CICS® Transaction Gateway for AIX®

Administration

Version 3.1

SC34-5591-00

CICS® Transaction Gateway for AIX®

Administration

Version 3.1

SC34-5591-00

Note!

Before using this information and the product it supports, be sure to read the general information under "Appendix B. Notices" on page 111.

First edition (September 1999)

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

This edition replaces SC34-5448-00. Material that was not in that book is indicated by vertical lines to the left of the material.

Contents

Figures

Tables

About this book

This book contains the following chapters:

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

Who should read this book

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

It is assumed that you are familiar with the operating system under which your CICS Transaction Gateway runs.

An understanding of Internet terminology would also be helpful.

Conventions and terminology used in this book

References to paths in this book use the OS/2® and Microsoft® Windows® convention of a backslash (\) as delimiter, instead of the / delimiter used on the AIX ® and Solaris platforms.

In this book, 'CICS on open systems' is used to refer to the following CICS server products, subject to availability:

- CICS for HP-UX

- CICS for Sun Solaris
- TXSeries for AIX
- TXSeries for HP-UX
- TXSeries for Solaris.

Prerequisite and related information

For information on the books available for this product, refer to "Appendix A. The CICS Transaction Gateway and CICS Universal Clients library" on page 105. 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. 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.

CICS Transaction Gateway is a robust and scalable complement to a Web server, and as such can be implemented as an e-business connector for IBM WebSphere™, which is a runtime environment for Java™ servlets.

The CICS Transaction Gateway is provided for the OS/2, Windows NT®, AIX, Solaris, and OS/390® platforms. The CICS Transaction Gateway is also provided for Windows 95/98, but on these platforms it can only be used for development purposes and not for production.

Figure 1 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. This is necessary only if you are using the CICS Transaction Gateway with Java applets.

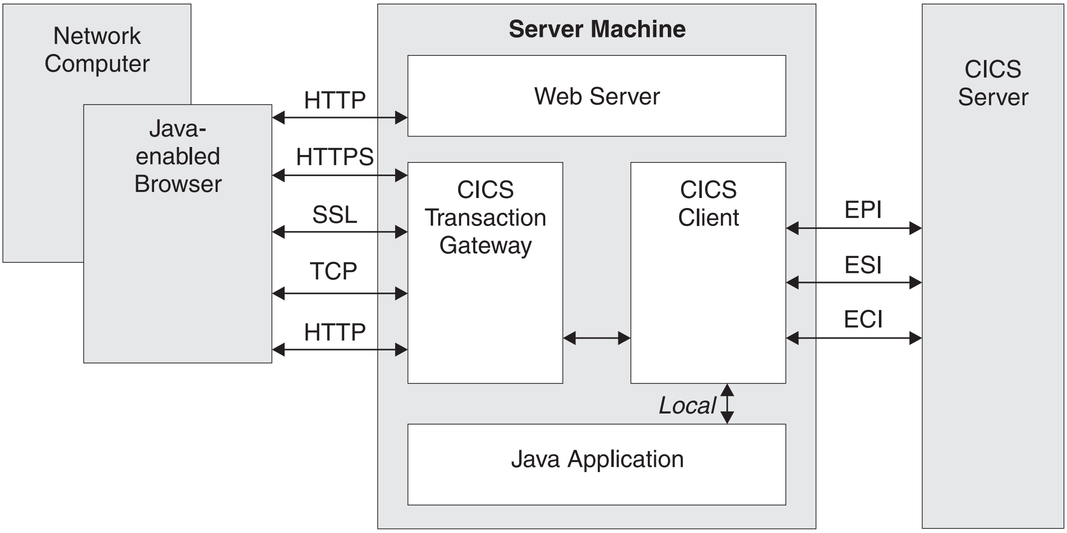


Figure 1. CICS Transaction Gateway

| 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 14).

What the CICS Transaction Gateway provides

The CICS Transaction Gateway provides the following:

1. A **Java gateway application** that is usually resident (for security reasons) on a Web server workstation. It communicates with CICS applications running in CICS servers through the ECI (External Call Interface), EPI (External Presentation Interface), or ESI (External Security Interface) interfaces provided by the CICS Universal Clients. This Java application was previously available in the IBM CICS Gateway for Java.
2. A **CICS Universal Client** that provides the ECI, EPI, and ESI interfaces, as well as terminal emulation function. The **ECI interface** enables a non-CICS Client application to call a CICS program synchronously or asynchronously as a subroutine. The **EPI interface** enables a non-CICS Client application to act as a logical 3270 terminal and so control a CICS 3270 application. The **ESI interface** enables a non-CICS Client application to invoke services provided by advanced program-to-program communication (APPC) password expiration management (PEM). The CICS Universal Clients allow communication with CICS Servers over the TCP/IP, APPC, NetBIOS, and other protocols, depending on the platform. See Table 2 on page 21 for more information.
3. 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 or servlet). The class **JavaGateway** is used to establish communication with the Gateway process, and uses the Java sockets protocol. The classes **ECIRequest**, **EPIRequest**, and **ESIRequest** are used to specify the ECI, EPI, and ESI calls respectively that are flowed to the gateway.
4. A **Terminal Servlet** that allows you to use a Web browser as an emulator for a 3270 CICS application running on any CICS server. The Terminal Servlet can be used with a Web server or a servlet engine that provides support equivalent to Java Servlet Development Kit (JSDK) Version 1.1 or later. This is an enhanced version of the function that was provided by the

CICS Internet Gateway in IBM CICS Clients Version 2. The Terminal Servlet provides an alternative to using the EPI programming interfaces.

5. A set of **Java EPI beans** for creating Java front-ends for existing CICS 3270 applications, without any programming.

The CICS Transaction Gateway can concurrently manage many communication links to connected Web browsers, and can control asynchronous conversations to multiple CICS server systems. The multithreaded architecture of the CICS Transaction Gateway 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 *servlets*, Java *applets,* and Java *applications.*

Java is an interpreted object-oriented language, similar to C++ which 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, include 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 (JVM).

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 servlets

Java servlets are small Java applications that run on a Web server machine, unlike Java applets, which are downloaded to a Web browser.

Java servlets have become popular as a replacement for CGI (Common Gateway Interface) programs. The advantage of a Java servlet over a CGI application is that they can execute more quickly since they are invoked as threads in a daemon process, meaning that they are persistent in memory and can fulfill multiple requests.

There are three ways in which you can run servlets:

Invoking the servlet by URL (Uniform Resource Locator)

The URL for a servlet typically has the following form:

`http://`*`machine-name`*`:`*`port`*`/servlet/`*`servlet-name`*

This method is used when you do not need the user to enter any information. Request parameters can be encoded in the URL, in the form of a query string.

Invoking the servlet with a form

This method involves creating an HTML form in a web page:

```
<FORM METHOD="GET" ACTION="/servlet/servlet-name">
attributes
</FORM>
```

This method is used when you need the user to enter information.

Invoking the servlet with a server-side include

A server-side include is processed by the Web server before the web page is sent to the user. In the HTML source, a server-side include looks like the following:

```
<SERVLET NAME="TerminalServlet" >
<PARAM NAME="request" VALUE="send">
<PARAM NAME="transaction" VALUE="CECI">
<PARAM NAME="display" VALUE="none">
</SERVLET>
```

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.

JavaBeans™

The JavaBeans API, developed by Sun Microsystems, allows you to write component software in Java. Components are self-contained, reusable software units that you can visually compose into applets or servlets using visual application builder tools, such as VisualAge® for Java. Any JDK Version 1.1-compliant browser or tool supports JavaBeans.

JavaBeans components are called beans. Most builder tools allow you to maintain beans in a palette or toolbox. You can select a particular Bean from the toolbox, drop it onto a form, modify its appearance and behavior, and define how it interacts with other beans. You can compose your selected Bean and other beans into an applet, servlet, or new Bean, and all this can be done without writing any code. For more information on JavaBeans, see the Sun Web site (www.java.sun.com).

The CICS Transaction Gateway provides **EPI Java beans** based on high-level EPI interfaces. These beans allow you to easily create Java programs (applets and applications) that access data from existing CICS 3270 applications, without any programming. Using a bean composer tool, such as VisualAge for Java, you can quickly and easily create new Java front-ends that can connect to CICS, run transactions, display data from 3270 screens, and send user input back to the CICS server. For more information, see the *CICS Transaction Gateway Programming* book.

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 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, see “Secure Sockets Layer (SSL)” on page 14 and “HTTPS” on page 15.

Web browsers and network computers

The CICS Transaction Gateway 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 20.

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) or servlets; 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, see “Web servers” on page 19.

How the CICS Transaction Gateway accesses CICS

This section describes how the CICS Transaction Gateway allows access to CICS programs and data.

For **Java applets**, access is achieved as follows:

1. The Web browser or network computer requests an HTML page from the Web server using the HTTP (Hypertext Transfer Protocol) protocol.
2. The Web server returns the HTML page containing a tag identifying a Java applet.
3. The browser starts requesting relevant Java classes from the Web server.
4. The Web server returns Java classes, including CICS Transaction Gateway classes as requested.
5. As classes are returned, the Java applet starts.
6. At some point the Java applet creates a **JavaGateway** object to connect to the CICS Transaction Gateway. This establishes communication between the browser and the long running Gateway process using Java’s sockets protocol.

7. The Java applet creates an **ECIRequest**, **EPIRequest**, or **ESIRequest** object containing respectively ECI, EPI, or ESI calls and sends it to the Gateway using the **JavaGateway.flow** method.
8. The Gateway receives the request, unpacks it, and makes corresponding ECI, EPI, or ESI calls to the CICS Universal Client.
9. The CICS Universal Client passes the ECI, EPI, or ESI calls to the intended CICS server.
10. The CICS server processes the call, including verification of the userid and password if required, and passes control and user data to the CICS application program.
11. When it has finished processing, the CICS application program returns control and data back to CICS, which passes it back to the Gateway.
12. The Gateway packs these results and returns them to the Java applet running on the Web browser.

For **Java servlets**, access is achieved as follows:

1. The Web server loads and initializes the servlet. This may be done when the Web server starts, or the first time a request is made to the servlet. The servlet may at this point create a **JavaGateway** object to connect to the CICS Transaction Gateway.
2. When the servlet is invoked by an appropriate HTTP request, the Web server calls its **Service** method with details of the request.
3. The Java applet creates an **ECIRequest**, **EPIRequest**, or **ESIRequest** object containing respectively ECI, EPI, or ESI calls and sends it to the Gateway using the **JavaGateway.flow** method.
4. The Gateway receives the request, unpacks it, and makes corresponding ECI, EPI, ESI calls to the CICS Universal Client
5. The CICS Universal Client passes the ECI, EPI, ESI calls to the intended CICS server.
6. The CICS server processes the call, including verification of the userid and password if required, and passes control and user data to the CICS application program.
7. When it has finished processing, the CICS application returns control and data back to CICS, which passes it back to the Gateway.
8. The Gateway packs these results and returns them to the Java servlet.
9. When the servlet receives the results of the requests it has made to the CICS Transaction Gateway, it generates a HTTP response to be returned to the Web browser.

Note: Not all of the ECI, EPI, and ESI calls are supported by **ECIRequest**, **EPIRequest**, and **ESIRequest** respectively. For more information, refer to *CICS Transaction Gateway Programming*.

The CICS Transaction Gateway: threading model

The CICS Transaction Gateway provides a multithreaded model for handling network connections, and assigning threads for requests from and 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 with the **Maximum number of Connection Manager threads** setting, see “Using the configuration tool” on page 30. You can also specify these limits when you start the CICS Transaction Gateway, see “Starting the Gateway with user-specified options” 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 an 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 30. You can also specify these limits when you start the CICS Transaction Gateway, see “Starting the Gateway with user-specified options” on page 71.

The threading model is illustrated in the following figures:

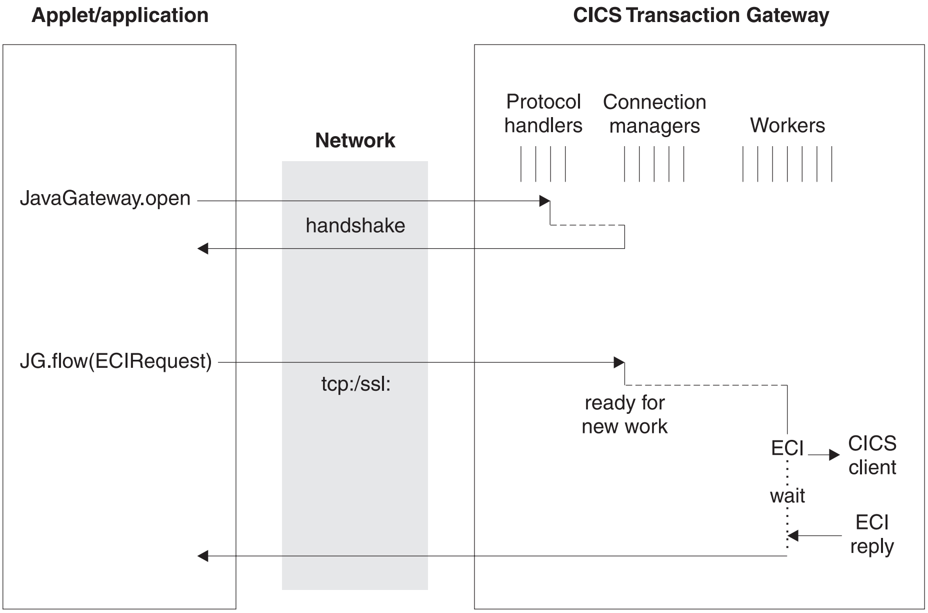


Figure 2. CICS Transaction Gateway threading model for tcp/ssl

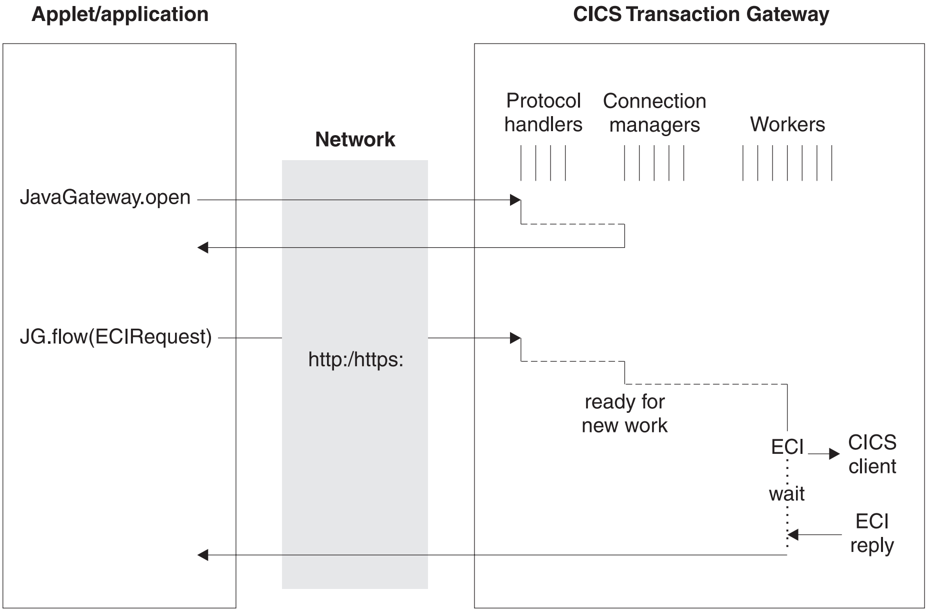


Figure 3. CICS Transaction Gateway threading model for http/https

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™ Task Control Blocks (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, 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 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.

In any of these cases, the choice between EPI and ECI is not always clear-cut, because both interfaces can be used to pass data between a non-CICS application and a CICS program. However, the mechanism is different in the two cases: 3270 data streams for EPI; application-defined formats in a COMMAREA for ECI.

External Presentation Interface (EPI)

The EPI allows a non-CICS application program to be viewed as a 3270 terminal by a CICS server system to which it is connected; both an EPI application and a CICS terminal can schedule transactions in a CICS server. The application can be using the facilities of several servers at the same time, and can act as if it were many different 3270 terminals. The application can schedule CICS transactions, and for these transactions it is the principal facility.

With CICS servers that support access through the EPI, other CICS transactions running in the server can use the CICS START command to schedule transactions that use the non-CICS application as their initiating terminal. When a non-CICS application uses the EPI to start a transaction in a CICS server, 3270 data streams and events are passed between the server and the application. The application can present the contents of the terminal I/O to its user in any manner appropriate to the application’s operating environment. Transactions can be routed to other CICS systems by standard transaction routing. Resources on other CICS systems can be accessed by function shipping.

Note that server transactions can be existing transactions that use 3270 input and output (with some restrictions).

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 CICS Transaction Server for OS/390 or CICS Transaction Server for VSE/ESA™ environment) by the front end programming interface (FEPI).

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

External Security Interface (ESI)

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

APPC PEM with CICS provides support for an APPC architected sign-on transaction that signs on user IDs to a CICS server and processes requests for a password change by:

- Identifying a user and authenticating that user’s identification
- Notifying specific users during the authentication that their passwords have expired
- Letting users change their passwords when (or before) the passwords expire
- Telling users how long their current passwords will remain valid
- Providing information about unauthorized attempts to access the server with a particular user identifier.

To use APPC PEM, the CICS Universal Client must be connected to the CICS server over APPC. An external security manager (ESM), such as resource access control facility (RACF), must also be available to the CICS server. ESI calls can be included within your ECI or EPI application. Only CICS servers returned by the **CICS_EciListSystems** and **CICS_EpiListSystems** functions are acceptable.

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. Security" on page 51; 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.

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.

The X.509 digital certificates are “encapsulated” into a Java classfile, which can then be used by the SSL and HTTPS protocols. These Java classfiles are referred to as *KeyRing* classes.

The CICS Transaction Gateway includes the iKeyman tool for the management of KeyRing classes. Using this tool, you can create KeyRing files, generate certificates, store certificates in KeyRings, and perform various other management functions. See “Chapter 5. Security” on page 51 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.

The security exit mechanism can also be used to authenticate/analyze an X.509 client certificate when client authentication is enabled.

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

CICS Transaction Gateway and object request brokers

Some Web servers contain an object request broker (ORB) that conforms to the Object Management Group’s (OMG) Common Object Request Broker (CORBA) standard. If you have such a Web server, you can develop servlet objects that can be invoked from a remote browser using the CORBA Internet InterORB Protocol (IIOP). IIOP is transmitted to the browser using HTTP.

Some browsers also include an ORB, which enables them to run programs that issue IIOP requests. This capability is used by Java applets that communicate by using the Java Remote Method Invocation (RMI), a binding mechanism that ultimately maps to the IIOP. With this technology an applet can communicate with servlets using IIOP and have the servlets invoke CICS transactions.

Other functions

Host on-Demand is supported, providing a means of accessing CICS 3270 applications via EPI on a Java-enabled Web browser. IBM eNetwork™ Host on-Demand Version 2.0, or later, is required.

Support is provided for a ***local CICS Transaction Gateway.*** This enables a Java program to communicate with a locally installed CICS Transaction

Gateway, without use of a network. Refer to “Preparing to use local CICS Transaction Gateway support” on page 50 for information on how to configure a local CICS Transaction Gateway.

Chapter 2. Planning before installation

This chapter helps you plan the installation of CICS Transaction Gateway by discussing the hardware and software requirements including the Web servers and browsers that are supported. The CICS servers to which CICS Transaction Gateway can connect are listed and possible migration issues are discussed.

Please refer to the product README file for any late changes to hardware and software requirements.

Hardware requirements

The CICS Transaction Gateway runs on any hardware capable of running the relevant operating system and other prerequisite software.

The hard disk requirement for your CICS Transaction Gateway (including documentation) is **23MB**.

Software requirements

The operating system requirements for your CICS Transaction Gateway are as follows:

- AIX Version 4.2 or later

Java Development Kit (JDK) Version 1.1.8, or later is supported.

Web servers

CICS Transaction Gateway has been tested with the following Web servers:

- Lotus Domino Go Webserver™ 4.6.2 or later for OS/2, Windows NT, AIX, or Solaris
- Microsoft Internet Information Server Version 4.0 for Windows NT, Windows 95, and Windows 98
- Sun Webserver Version 2.0 for Solaris
- IBM HTTP Server (Apache) Version 1.3 for Windows NT, AIX, and Solaris
- WebSphere Advanced Server for OS/390 (included in OS/390 V2 R8)
- Netscape Enterprise Server Version 3.6 for Windows NT, AIX, and Solaris

If you are going to run the terminal servlet supplied with CICS Transaction Gateway, you will need a Web server or a servlet engine that provides servlet support equivalent to Java Servlet Development Kit (JSDK) Version 1.1 or later, for example:

- IBM WebSphere Application Server, Advanced or Enterprise Edition, Version 2.02 or later for Windows NT, AIX, or Solaris.

Note: Domino Go WebServer Version 4.6.2 does not support JSDK 1.1

Web browsers

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

- OS/2:Netscape Version 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

Universal CICS Clients

CICS Transaction Gateway incorporates CICS Universal Clients Version 3.1. You cannot use CICS Clients Version 2 with CICS Transaction Gateway.

The CICS Universal Clients can communicate with CICS servers using a number of protocols:

TCP/IP Transmission Control Protocol/Internet Protocol (TCP/IP) is supported on all platforms.

APPC Advanced Program-to-Program Communication (APPC) is supported on all platforms.

TCP62 TCP62 is supported on the OS/2 and Windows platforms, this allows APPC communication over a TCP/IP network.

NetBIOS Network Basic Input/Output System (NetBIOS) is supported on the OS/2 and Windows platforms.

Refer to the *CICS Universal Client for AIX Administration* book for more information on protocols, communication products that support these protocols, and also application development language support.

CICS servers

A CICS server runs real-time multi-user applications and manages the associated resources and data. CICS servers can run on a range of platforms, from workstations to highly parallel mainframe systems.

CICS Transaction Servers, and TXSeries Servers, contain CICS servers and CICS Clients, and can contain system management products.

For more general information see the CICS Web site:

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

The CICS Universal Client component of the CICS Transaction Gateway can connect to CICS servers using a variety of protocols. Table 2 shows the protocols available to CICS Universal Client for AIX for connecting to CICS servers.

Table 2. Availability of protocols for connecting to CICS servers

CICS Server	**Protocol**
CICS for MVS/ESA™ Version 4 Release 1	APPC
CICS Transaction Server for OS/390 Version 1 Release 2, and later	APPC
CICS for VSE/ESA Version 2 Release 3	APPC
CICS Transaction Server for VSE/ESA Version 1 Release 1	APPC
CICS Transaction Server for OS/2 Warp, Version 4.1, and later	TCP/IP, APPC
Transaction Server for Windows NT Version 4, TXSeries Version 4.2 for Windows NT	TCP/IP, APPC
‘CICS on Open Systems’ (see “Conventions and terminology used in this book” on page ix for a definition)	TCP/IP, APPC
CICS for OS/400® Version 4 Release 4 and later	APPC

For more information on connectivity between CICS Universal Clients and CICS servers, refer to the *CICS Universal Client for AIX Administration* book.

CICS server PTF requirements

For signon capable terminals, you must ensure that you have the appropriate PTF level applied to any CICS servers being used. You should refer to the CICS Transaction Gateway / CICS Universal Client README file for the latest details and check the PTFs for the CICS servers.

To provide complete support for timeouts, if you are using any TXSeries or Transaction Server on UNIX® and Windows NT platforms, it must include the appropriate PTF level. You should refer to the CICS Transaction Gateway / CICS Universal Client README file for the latest details and check the PTFs for the CICS server.

Migration issues

This section discusses migration issues that you will encounter if you previously used a product related to CICS Transaction Gateway.

Migrating from CICS Universal Client Version 3.0

Before you install CICS Transaction Gateway, you may already have installed CICS Universal Clients Version 3.0. You may have customized versions of the following configuration files:

File	**Default Name**
Keyboard mapping file	CICSKEY.INI
Color mapping file	CICSCOL.INI
Windows settings	CICSTERM.INI

When you install CICS Transaction Gateway your customized files are preserved, even though the old CICS Universal Client Version 3.0 is deleted.

In CICS Transaction Gateway Version 3.1 the configuration file, (CTG.INI), is used instead of the client initialization file (CICSCLI.INI). To convert the client initialization file, you can run the configuration conversion tool (ctgconv), see “The configuration conversion tool” on page 45.

Migrating from CICS Transaction Gateway Version 3.0

In CICS Transaction Gateway Version 3.1 the configuration file, (CTG.INI), is used instead of the Gateway.properties file. To convert the file, you can run the configuration conversion tool (ctgconv), see “The configuration conversion tool” on page 45.

Migrating from CICS Gateway for Java

If you are a user of CICS Gateway for Java migrating to CICS Transaction Gateway, you may have a customized version of the Gateway.properties file. To convert this file, you can run the configuration conversion tool (ctgconv), see "The configuration conversion tool" on page 45.

When you install over CICS Clients Version 2, the CICS Gateway for Java is not automatically deleted, so you will need to delete it.

Due to the renaming of the CICS Transaction Gateway 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.epi.*      to: import com.ibm.ctg.epi.*

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

Programs written for the CICS Gateway for Java prior to Version 2.0.1 that use the Session interface need to be modified because the parameter to the Session.handleReply method was changed to TerminalInterface instead of a Terminal object. Existing programs must be changed then recompiled.

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 Internet Gateway

The functionality of the CICS Internet Gateway component of CICS Clients Version 2 has been incorporated in CICS Transaction Gateway as a Java Terminal servlet, rather than a CGI application.

Migration from the CICS Internet Gateway is described in "Migrating from the CICS Internet Gateway" on page 95.

Chapter 3. Installing CICS Transaction Gateway

This chapter describes how to install CICS Transaction Gateway for AIX.

Installing CICS Transaction Gateway on AIX

The CICS Transaction Gateway is distributed as a compressed tar file containing the library files, commands, messages and customization files (.INI) in a single directory tree.

To install CICS Transaction Gateway on AIX

1. Log in as root.
2. Stop any CICS Universal Client that is running (cicscli -i), and run the slibclean command to remove any unused libraries.
3. Find the tar file on the distribution medium. It is called ctg-310x.tar.Z.
4. Copy the compressed tar file ctg-310x.tar.Z to a working directory on your hard disk.
5. Uncompress the file and extract the files:

   ```
   uncompress ctg-310x.tar
   tar -xvf ctg-310x.tar
   ```

   The files will be extracted to the /usr/lpp/ctg directory.

   Samples are provided in the samples subdirectory.
6. Set up the CICS Transaction Gateway by entering:

   ```
   mkcicscli
   ```

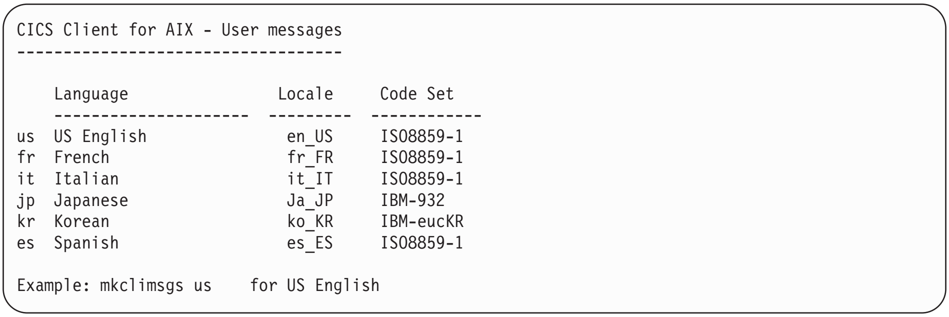
Note: You can uninstall CICS Transaction Gateway by using the `mkcicscli -u` command.

National language support

You can select the language in which user messages are displayed by CICS Transaction Gateway by entering the command:

```
mkclimsgs XX
```

where *XX* is the message language. To obtain a list of the available languages, enter the command without the *XX* parameter:

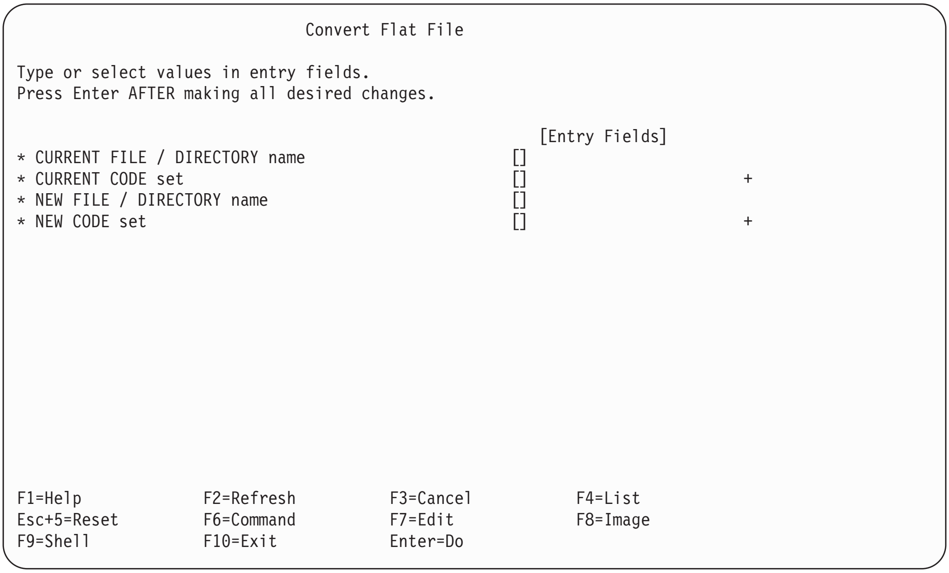


```
CICS Client for AIX - User messages
-----------------------------------

    Language                Locale     Code Set
    ---------------------  ---------  ------------
us  US English               en_US     ISO8859-1
fr  French                   fr_FR     ISO8859-1
it  Italian                  it_IT     ISO8859-1
jp  Japanese                 Ja_JP     IBM-932
kr  Korean                   ko_KR     IBM-eucKR
es  Spanish                  es_ES     ISO8859-1

Example: mkclimsgs us    for US English
```

This also shows the locale and code set associated with the language.

To use an alternate code set, you can use the iconv routine (smitty iconv) for the flat file /usr/lpp/ctg/CCLMSG.TXT.

The Convert Flat File screen is as follows:



```
                               Convert Flat File

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
* CURRENT FILE / DIRECTORY name                      []
* CURRENT CODE set                                   []                       +
* NEW FILE / DIRECTORY name                          []
* NEW CODE set                                       []                       +










F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Fill in this screen as follows:

CURRENT FILE / DIRECTORY name
Enter /usr/lpp/ctg/CCLMSG.TXT

CURRENT CODE set
Enter the code set for the language you selected with mkclimsgs.

NEW FILE / DIRECTORY name
Enter the name of the new file.

NEW CODE set

Enter the alternative code set. You can view a list of the alternative code sets using the smitty lang command, which is used to set the language environment. For example, the code set IBM-850 is an alternative for the US English code set ISO8859-1.

When you have done the conversion, overwrite the CCLMSG.TXT file with the new file.

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.

Chapter 4. Configuring CICS Transaction Gateway

This chapter describes the following:

- "Configuring your programming environment for CICS Transaction Gateway"
- "Using the configuration tool" on page 30
- "The configuration conversion tool" on page 45.
- "Editing the configuration file" on page 47.
- "Configuring the client keyboard mapping" on page 49.
- "Configuring the client screen colors and attributes" on page 49.
- "Preparing to use local CICS Transaction Gateway support" on page 50.

Configuring your programming environment for CICS Transaction Gateway

The Java Virtual Machine (JVM) uses the CLASSPATH environment variable to find classes, and zip or jar archives containing classes. To allow the JVM to access class files, you must specify the full path of directories containing class files or archives.

To compile and run Java applications for use with the CICS Transaction Gateway, add the full path of the following to the CLASSPATH environment variable:

- ctgclient.jar
- ctgserver.jar (if you are using a local gateway)

These archives are in the classes subdirectory of the directory where you installed the CICS Transaction Gateway.

Setting CLASSPATH on AIX

To set the CLASSPATH environment variable, use a shell command like the following (assuming the ksh shell):

```
export CLASSPATH=.:/usr/local/classes:/usr/lpp/ctg/classes/ctgclient.jar
```

This shell command makes the classes in the current directory, the directory /usr/local/classes, and in archive /usr/lpp/ctg/classes/ctgclient.jar accessible by the JVM.

You may want to place the shell command in the .profile file in your home directory so that the CLASSPATH environment variable is automatically set when you log on to your system.

To display the value of the CLASSPATH environment variable use the following shell command:

```
echo $CLASSPATH
```

Using the configuration tool

You use the configuration tool to set configuration parameters for the CICS Transaction Gateway and CICS Universal Clients.

To use the configuration tool you must export the display using the commands described in “Using X-Windows from a remote system” on page 27.

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

When you start the configuration tool for the first time, a number of TaskGuides help you in setting up a new configuration. The TaskGuides let you:

1. Create CICS server definitions, and define the protocols to be used by the CICS Universal Clients
2. Decide which protocols to use with the CICS Transaction Gateway

Default settings for the CICS Universal Clients are created.

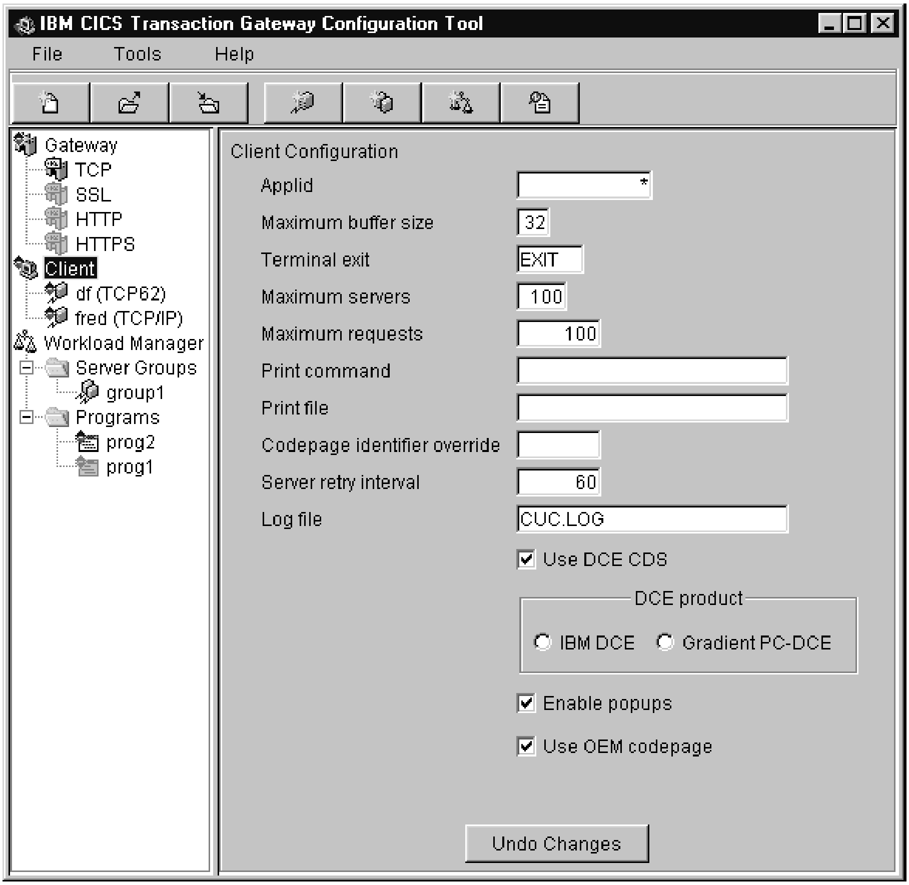
The configuration is stored by default in the CTG.INI file in the bin subdirectory where you installed the CICS Transaction Gateway. You can edit this file directly, but it is recommended that you use the configuration tool to perform configuration.

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

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.



Client Configuration

Applid: *

Maximum buffer size: 32

Terminal exit: EXIT

Maximum servers: 100

Maximum requests: 100

Print command:

Print file:

Codepage identifier override:

Server retry interval: 60

Log file: CUC.LOG

☑ Use DCE CDS

DCE product: ○ IBM DCE ○ Gradient PC-DCE

☑ Enable popups

☑ Use OEM codepage

Figure 4. The configuration tool

Note: On your platform, the configuration tool may not appear exactly as shown in Figure 4.

Tree structure

The tree structure (see Figure 4) allows you to navigate through all of the settings in your configuration. The types of root node are as follows:

Gateway Contains up to four subnodes, that is, one for each protocol the CICS Transaction Gateway can use (TCP, SSL, HTTP, and HTTPS).

Client Contains a subnode for each of your server definitions.

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 options:

Trace Settings Displays the Trace Settings dialog.

New Server Starts the TaskGuide for creating new server definitions.

Delete Server Deletes a server node from the tree structure.

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 that 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

To display the Gateway Settings panel, select the Gateway node in the tree structure. The settings map to the parameters in the Gateway section of the CTG.INI file.

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 SSL, the default is 8050.

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.

SSL protocol settings

Select the **SSL** subnode to display the settings for 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 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. Security” on page 51.

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 61.

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.

HTTPS protocol settings

Select the **HTTPS** subnode to display the settings for HTTPS. 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.

Configuring Client settings

To display the Client Settings panel, select the Client node in the tree structure. The settings map to the parameters in the Client section of the CTG.INI file.

Applid

Enter up to 8 characters, or leave this field with the default value of *.

This value specifies the applid of the CICS Universal Client workstation in the form in which it will be autoinstalled as a system at the CICS server. The name must be unique within the CICS server system. The value of * automatically generates a name that is guaranteed to be unique.

Note: If the client is to be autoinstalled to more than one CICS server, and if you enter a specific name for the applid, that name must be unique with respect to all servers it is connected to. If the name is not unique, then attempts to connect to a server may be rejected because another client has already been installed using the same name. If a name of * is used, the client may be known by a different unique name at each server.

If the client is to communicate with a given server via APPC, then this applid may be overridden at the time the client is installed at the server by the Local LU name for the client.

Maximum buffer size

Enter a number of kilobytes, in the range 4 through 32. The default is 32 KB.

This value specifies the size of the transmission buffers in which application or terminal data will flow. The value should be large enough to cater for the largest possible COMMAREA or terminal input/output area (TIOA) to be used. The value does *not* include an overhead of 512 bytes needed by the clients for some protocols.

This setting need normally be specified only for clients running in a memory-constrained environment.

Terminal exit

Enter a character string of between 1 and 4 characters. The default is EXIT.

The string, when entered at a terminal emulator at any time and place where a transaction name can be entered, causes the terminal emulator to terminate. The string must not contain any blank characters.

The string is case-sensitive. If a terminal emulator has uppercase translation in its CICS terminal definition, you should enter this string in uppercase.

Maximum servers

Enter a value in the range 1 through 256. The default is 10.

This value specifies the maximum number of servers that can be accessed concurrently from the client.

Maximum requests

Enter a value in the range 1 through 10 000. The default is 90.

This value specifies the maximum number of concurrent items that may be executing on the client, when an item is defined as a terminal emulator, an EPI terminal or an ECI unit of work.

This value is used to detect runaway situations where an application could, in error, submit an excessive number of requests to a server. The actual limit may be less than this setting if other operating system limits (for example, memory constraint or communication sessions), come into effect.

Print command

Enter a character string, from 1 to 256 characters long.

The specified string is a command specific to the operating system under which the client is running. When a request to print is received at the client, the client generates a temporary print file with a unique name per print request.

The parameter string is appended with the temporary file name, and the resultant command executed. This allows, for example, print requests to be copied to a file, directed to a local printer, formatted for inclusion into documentation, and so on.

It is the responsibility of the Print Command to delete the temporary print file after it has finished processing it.

Use a shell script with the print command (for example, **lpr**) followed by the command to delete (**rm**).

See also the **Print file** description for more information.

Print file

Enter a character string, 1 to 256 characters long.

This option is applicable only if the Print Command setting is omitted.

The specified string identifies a file to which output from print requests received at the client is directed. Each print request is appended to the end of the current file.

Note: This setting acts only as a default. The terminal and print emulators provide options to override this value.

Codepage identifier override

Enter a value indicating a Coded Character Set Identifier (CCSID) to override your local codepage identifier.

You should use this setting if your platform has been updated for Euro support, and the CICS Server has Euro support. For example, for Latin-1 countries, use a CCSID value of 858 to indicate that the codepage 850 includes Euro support. For codepage 1252, specify a CCSID value of 5348.

Note that cicsterm will always display characters based on the local codepage of the workstation, regardless of the value specified by the **Codepage identifier override** setting.

Also note that if you use the CCSID to change the codepage identifier used, data that was stored previously in the server may be modified when retrieved to the Client, if it includes characters for which the code points produce different characters.

Server retry interval

Enter a number of seconds. This is the time in seconds between attempts by the client to reconnect to a server to which it was connected. The default interval is 60 seconds.

When the client becomes aware that a server it was connected to is no longer active, it attempts to reconnect to the server 1 second after it becomes inactive. Subsequent retry attempts then occur at the interval defined by **Server retry interval**.

Log file

Enter the name of the log file to be used for problem diagnosis.

If not specified, the log filename defaults to CICSCLI.LOG in the /var/cicscli subdirectory.

Configuring Server settings

To display the Server Settings panel, select a Server node in the tree structure. The settings map to the parameters in a Server section of the CTG.INI file.

Server name

Enter a name of between 1 and 8 characters. This provides a communications-protocol-independent name for the server, local to the client.

Requests to access the server from ECI, EPI, ESI, or terminal emulators reference the server through this name.

Description

Enter a description for the server of between 1 and 60 characters. This description is optional.

The description is returned to applications running on the client via the CICS_EpiListSystems and CICS_EciListSystems functions. (See the *CICS Family: Client/Server Programming* book).

Initial transaction

Enter a transaction identifier of between 1 and 128 characters.

This string is case-sensitive and identifies the initial transaction (and any parameters) to be run when the terminal emulator connects to the server. If you do not enter anything, no initial transaction is run. The first four characters, or the characters up to the first blank in the string are taken as the transaction. The remaining data is passed to the transaction on its invocation.

Model terminal definition

Enter a string of between 1 and 16 characters.

The string is case-sensitive and specifies the name of a model terminal definition at the server, identifying the characteristics of terminals to be autoinstalled from the client. If the model cannot be located at the server, or you do not enter anything, a default terminal definition is used. This default is server-specific.

The interpretation of the Model terminal definition setting is server-specific. For example, for a CICS for OS/2 server, the value is validated (by the server) as a 1- to 4-character string identifying a model entry in the CICS terminal definition control tables with a terminal name set to the value specified on this parameter. For a TXSeries for AIX server, the value is 1 to 16 characters, and is the DevType for a CICS terminal definition entry to be used as the model.

Use upper case security

Select this check box to specify that any userid or password from an ECI application or resulting from a user prompt is converted to uppercase.

This setting is disabled by default.

TCP/IP

Select this option to specify TCP/IP as the protocol for the server connection.

SNA

Select this option to specify SNA as the protocol for the server connection.

Protocol settings displayed on the panel change according to the protocol button you select.

TCP/IP settings

To display these settings, select the **TCP/IP** option.

Hostname or IP address: Enter the character or numeric TCP/IP identifier for the host on which the CICS server is running. For example, cicssrv2.company.com (HostName) or 9.20.4.1 (IPAddress).

Hostnames are mapped to IP addresses either by the name server or in the hosts file in the etc subdirectory. It is, however, better to use a Hostname in case the IP address changes.

Port: Enter a numeric value in the range 0 through 65 535 defining the port number at the server to which the client should connect. The default value is 0.

A value of 0 indicates that the services file in the TCP/IP /etc directory should be used to locate the port number for the service CICS using a protocol of TCP.

If no entry can be located in the services file, a value of 1435 is assumed, this being the TCP/IP architecture assigned port for CICS Universal Clients.

Connection timeout: Enter a value in the range 0 through 3600, specifying the maximum time in seconds that establishing a connection is allowed to take; the default value of 0 means that no limit is set by the client.

A timeout occurs if connection establishment takes longer than the specified time. The TCP/IP socket is closed and the return code passed back to the client application is either ECI_ERR_NO_CICS or CICS_EPI_ERR_FAILED.

Send TCP/IP Keepalive packets: Select this check box if you want TCP/IP to periodically send KeepAlive packets to the server to check the connection.

SNA settings

To display these settings, select the **SNA** option.

Use LU alias names: Select this check box to use LU alias names.

LU names are always aliases on AIX and Solaris, so this setting is ignored for these clients.

Partner LU name: Enter the LU Name of the server as it is known to the APPC configuration at the CICS Universal Client.

This must be an eight-character alias name.

Local LU name: Enter the name of a local LU to be used when connecting to the server. The same LU can be used for all server connections.

This must be an eight-character alias name.

Mode name: Enter between 1 and 8 characters specifying the mode name to be used when connecting to the server. Omitting this setting results in a blank mode name.

Trace settings

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

Trace Settings

Select check boxes to specify the CICS Transaction Gateway and CICS Universal Client components that will be traced when tracing is turned on.

Trace everything	All components.
Client API level 1	The client API layer (level 1).
Client API level 2	The client API layer (level 1 and 2).
CICSCLI command line	The cicscli command interface.
CICSTELD	cicsteld process.
CICSTERM and CICSPRINT	cicsterm and cicsprnt emulators.
CPP classes	The C++ class libraries.
Client daemon	CICS Universal Client daemon.
Transport layer	Interprocess communication.
Protocol drivers	Protocol drivers (TCP, SNA, and so on).
Java Gateway	The CICS Transaction Gateway. If you only have a CICS Universal Client, you will not see this setting.

You can also specify trace components by using the -m parameter on the cicscli command (except the Java Gateway component). Any component tracing specified using cicscli overrides that specified with the configuration tool. If component tracing is not specified either by the cicscli command or the configuration tool, a default set of components is traced, namely: **Protocol drivers, Client daemon**, and **Client API level 1**. If you select any of the check boxes, this overrides the default set of components.

For the API component, you can specify the level of information to trace. The **Client API level 1** check box specifies that basic API-related information is traced, for example, before and after ECI, EPI, and ESI calls. The **Client API level 2** check box specifies that additional API trace entries are produced in addition to those of level 1.

Note that the cicscli -d=*nnn* command is used to set the maximum size of the data areas to be traced. The trace data may be truncated if you set *nnn* lower than the size of data expected.

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.

Client trace file

Enter the pathname of a trace file to which trace messages will be written, if tracing is enabled.

You do not have to enter an extension for the filename, as a file of type .BIN is always generated (or .WRP if the trace file wraps).

If no path is specified, the trace is written to the CICSCLI.BIN file. This is in the /var/cicscli subdirectory.

You can convert the binary trace file to an ASCII file using the cicsftrc command.

Maximum Client wrap size

Enter a value in the range 0 through 999 kilobytes. The default is 0.

This value specifies how large the trace file will grow on disk before wrapping. If the default value of 0 is specified, wrapping of trace is disabled.

The configuration conversion tool

You use the configuration conversion tool (ctgconv) to convert the configuration files of previous versions of CICS Transaction Gateway, IBM CICS Universal Clients, and CICS Gateway for Java to the new format of the CICS Transaction Gateway Version 3.1 configuration file.

The conversion tool converts the following:

Gateway.properties	Properties file of CICS Transaction Gateway Version 3.0 and CICS Gateway for Java.
CICSCLI.INI	Client initialization file of CICS Clients Version 2 and Version 3.0.

The conversion tool produces one output file called CTG.INI by default. Samples from this file are shown in "Editing the configuration file" on page 47.

The old files are 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 -c=file [-g=file]  [-o=file]
```

for each parameter, *file* can be:

- a filename with extension, in which case the /usr/lpp/ctg/bin directory is assumed. The extension must be .INI, except for the CICS Transaction Gateway input file, which has the .properties extension.
- a full pathname.
- a directory name (without / at the end), in which case the default filename is assumed.

-c=*file*
: Specifies, the client initialization file to be converted. This parameter is mandatory.

  If *file* specifies a directory, a filename of CICSCLI.INI is assumed.

  If the parameter is not specified, the CICSCLI environment variable is used to locate the client initialization file. If CICSCLI is not set, the file /usr/lpp/ctg/bin/CICSCLI.INI is assumed.

-g=*file*
: Specifies the Gateway.properties file to be converted.

  If you do not specify the **-g** parameter, no Gateway section is created in CTG.INI.

-o=*file*
: Specifies the pathname of the converted file. The default is /usr/lpp/ctg/bin/CTG.INI.

To get help on using ctgconv, enter: `ctgconv -?`

During conversion, the input files are renamed so that the last three characters of the file extension become .BAK. Any file with that name is overwritten, and any output file that exists before the conversion is renamed in the same way.

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.

The configuration file is used for both CICS Universal Clients and CICS Transaction Gateway, and contains the following sections:

1. GATEWAY (CICS Transaction Gateway only)
2. CLIENT
3. SERVER
4. DRIVER

These sections have the format:

```
SECTION sectioname [=value]
     property1=value
     property2=value
     ...
     propertyN=value
ENDSECTION
```

You must restart both the CICS Universal Client and the CICS Transaction Gateway to pick up any changes to the configuration file.

The following sections give examples of the properties in each section. For information on the properties, refer to the descriptions of the corresponding settings in the configuration tool.

GATEWAY section

```
SECTION GATEWAY
# -------------------------------------------------------------------------------
# Properties file for the CICS Transaction Gateway
# -------------------------------------------------------------------------------

        initconnect=1   # Initial number of ConnectionManager threads
        maxconnect=100  # Maximum number of ConnectionManager threads
        initworker=1    # Initial number of Worker threads
        maxworker=100   # Maximum number of Worker threads
        trace=on        # Enable extra tracing messages
        notime=on       # Disable timing information in messages
        noinput=on      # Disable the reading of input from the console
        nonames=on      # Do not display TCP/IP hostnames

# -------------------------------------------------------------------------------
# Entries to define the protocols which this CICS Transaction Gateway daemon will
# accept.
#
# Entries for a particular protocol begin with protocol@<protocol-name>.
# followed by which property is being set. The following properties MUST be set
# for each protocol handler to be used :
```

```
#
#  (1) protocol@<protocol-name>.handler = ....
#         The name of the class which provides the handler for this protocol
#
#  (2) protocol@<protocol-name>.parameters = ....
#         The parameters passed to the protocol handler. The possible parameters
#         vary upon the protocol handler.
#

        protocol@tcp.handler=com.ibm.ctg.server.TCPHandler
        Protocol@tcp.parameters=port=2006;sotimeout=1000;
        connecttimeout=2000;idletimeout=600000;pingfrequency=60000

        protocol@http.handler=com.ibm.ctg.server.HttpHandler
        Protocol@http.parameters=port=8080;sotimeout=1000;connecttimeout=2000;
        idletimeout=120000;requiresecurity

        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=off;

        Protocol@https.handler=com.ibm.ctg.server.HttpsHandler
        Protocol@https.parameters=port=443;sotimeout=1000;
        connecttimeout=2000;Idletimeout=120000;keyring=ServerKeyRing;keyringpw=default;
        clientauth=off;

# -----------------------------------------------------------------------------
# Advanced settings.

        ecigenericreplies=off
        workertimeout=10000
        closetimeout=10000

ENDSECTION
```

CLIENT section

```
SECTION CLIENT = *
#-----------------------------------------------------------------------
# Client section - This section defines the local CICS client.  There
#                  should only be one Client section.

        MaxServers = 1                  # Only allow one server connection
        MaxRequests = 256               # Limit the maximum server interaction
        MaxBufferSize = 32              # Allow for a 32K maximum COMMAREA
#       CCSID=850

ENDSECTION
```

SERVER section

```
SECTION SERVER = cicssna1
#-----------------------------------------------------------------------
# Server section - This section defines a server to which the client may
#                  connect.  There may be several Server sections.
```

```
        Description = A SNA Server      # Arbitrary description for the server
        Protocol = SNA                  # Matches with a Driver section below
        NetName = ABCDEFGH              # The server's Alias Name
        LocalLUName = WXYZ9999          # The client's local LU name alias
        ModeName = LU62PS               # The SNA communications mode name

ENDSECTION
```

DRIVER section

```
SECTION DRIVER = SNA            ; TCPIP, SNA
#-----------------------------------------------------------------------
#       Driver section - This section defines a communications protocol DLL
#                used to communicate with a server.  There may be
#                several Driver sections.

        DriverName = CCLIBMSN           #CCLIBMSN, etc.

ENDSECTION
```

Note: The DRIVER section does not correspond to any settings in the configuration tool. The configuration tool selects the correct protocol drivers automatically.

Configuring the client keyboard mapping

The keyboard mapping for terminal emulator operation is defined in a file called CICSKEY.INI by default. A sample mapping file is supplied, but it is recommended that you create your own customized mapping file.

You can identify which keyboard mapping file is used with the environment variable CICSKEY, or with the -f parameter of the cicsterm command.

For information on configuring the keyboard mapping file, refer to the *CICS Universal Client for AIX Administration* book.

Configuring the client screen colors and attributes

A color mapping file is used to provide alternative representations in hardware environments where it is not possible to exactly replicate 3270 screen attributes, for example, blinking or underscore. The color mapping file therefore defines how 3270 screen attributes are emulated on the client hardware.

The screen colors and attributes are defined in a file called CICSCOL.INI by default. A sample mapping file is supplied, but it is recommended that you create your own customized mapping file.

You can identify which color mapping file is used with the environment variable CICSCOL, or with the -c parameter of the cicsterm command.

For information on configuring the color mapping file, refer to the *CICS Universal Client for AIX Administration* book.

Preparing to use local CICS Transaction Gateway support

Before you can run an application that uses the **local** CICS Transaction Gateway support, you must ensure that your environment is correctly configured. The following requirements must be met:

1. **A correctly configured Java environment**

   On AIX, you need a correctly configured Java environment. However, any Java application that uses a **local** CICS Transaction Gateway object to call CICS **must** be executed using the supplied cicsjava script; cicsjava can be found in the CICS Transaction Gateway's bin subdirectory and has syntax identical to that of the standard java executable. The cicsjava script will correctly perform the environment settings and so no further setup is required.

2. **Correct settings to allow the CICS Transaction Gateway binaries to be found**

   The CICS Transaction Gateway .class files need to be available. You should set your CLASSPATH environment variable to include the CICS Transaction Gateway's ctgserver.jar and ctgclient.jar archives. Also, the relevant binaries must be available.

   You should set your PATH environment variable to include CICS Transaction Gateway's bin subdirectory so that the cicsjava script can be called easily. All other settings are made by the cicsjava script.

To run the application: use the supplied cicsjava command.

Chapter 5. Security

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

- “Overview” 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 55 describes the SSL protocol and the types of authentication it provides.
- “The ctgikey tool” on page 57 introduces iKeyMan, the tool provided by CICS Transaction Gateway for managing your digital certificates.
- “Using externally-signed certificates (SSLight)” on page 58 describes how to obtain externally-signed digital certificates and store them in KeyRing files for use by the CICS Transaction Gateway.
- “Using self-signed certificates (SSLight)” on page 63 describes how to generate self-signed digital certificates for use by the CICS Transaction Gateway
- “Configuring CICS Transaction Gateway for SSL and HTTPS” on page 67 describes how to configure CICS Transaction Gateway to use the secure protocols.

The implementation of SSL provided by CICS Transaction Gateway is written in pure Java, and is referred to as **SSLight**. Another implementation, **System SSL** applies only to the CICS Transaction Gateway on OS/390 and can only be used for SSL servers on that platform.

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. 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.

Obtaining an externally-signed certificate is described in "Using externally-signed certificates (SSLight)" on page 58.

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.

Generating a self-signed certificate is described in “Using self-signed certificates (SSLight)” on page 63.

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.

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 SSL and HTTPS” on page 67.

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 5.

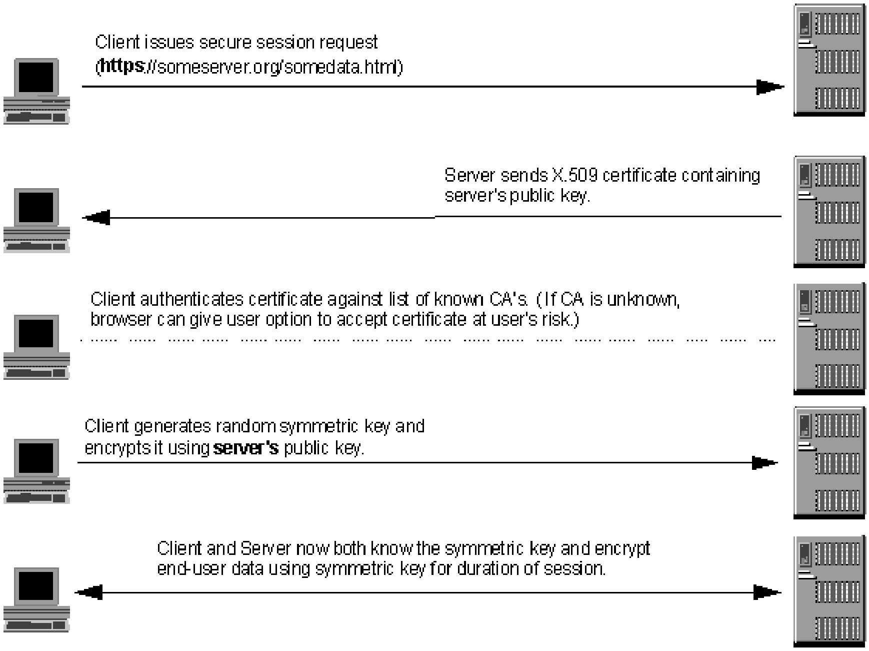

Client issues secure session request
(https://someserver.org/somedata.html)
Server sends X.509 certificate containing
server's public key.
Client authenticates certificate against list of known CA's. (If CA is unknown,
browser can give user option to accept certificate at user's risk.)
Client generates random symmetric key and
encrypts it using server's public key.
Client and Server now both know the symmetric key and encrypt
end-user data using symmetric key for duration of session.

Figure 5. 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 58.
- Generate self-signed certificates, see “Using self-signed certificates (SSLight)” on page 63.
- 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 61.

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.

Using externally-signed certificates

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.

Using self-signed certificates

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.

Configuring CICS Transaction Gateway for SSL and HTTPS

The CICS Transaction Gateway supports various protocols for communicating with its clients, including TCP, HTTP, 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 30). The SSL and HTTPS protocols will then be started when the CICS Transaction Gateway is executed. You can specify that just one of the security protocols is used, or both. With these handlers specified, when the CICS Transaction Gateway is started, it listens for SSL requests on port 8050, and for HTTPS requests on port 443.

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

KeyRing classname
This setting specifies the name of the 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 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 classfile 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 operation

This chapter describes:

- “Starting the Gateway”.
- “Stopping the Gateway” on page 73.

Starting the Gateway

You start the Gateway at the operating system command prompt of the computer on which you have installed it.

First, set your working directory to the CICS Transaction Gateway’s bin directory.

You can start the CICS Transaction Gateway in two ways:

- “Starting the Gateway with preset options”
- “Starting the Gateway with user-specified options”.

Starting the Gateway with preset options

To start the Gateway with predefined options: Type `ctgstart` at the command prompt and press Enter.

When you start the Gateway in this way, the default configuration settings, or those you configured using the configuration tool, are used. (see “Using the configuration tool” on page 30).

A Gateway console session is started, and the following message is displayed:

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

This is followed by lines showing the values being used:

```
CCL6502I: [ Initial ConnectionManagers = 1, Maximum ConnectionManagers = 100,
CCL6502I: Initial Workers = 1, Maximum Workers = 100, Port = 2345 ]
```

Starting the Gateway with user-specified options

The user definable options on the start command are:

```
-port=port_number
-initconnect=number
-maxconnect=number
-initworker=number
```

```
-maxworker=number
-trace
-notime
-tfile=pathname
-noinput
-nonames
```

where:

-port
Specifies a TCP/IP port number for the tcp: protocol.

-initconnect
Specifies an initial number of ConnectionManager threads.

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

-initworker
Specifies an initial number of Worker threads.

-maxworker
Specifies a maximum number of Worker threads. If this value is set to -1, no limits are applied to the number of ConnectionManager threads.

-trace
Enables extra tracing messages (see “Using trace” on page 101.)

-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
Disables display of TCP/IP hostnames.

-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 start-up options you require, and press Enter. Options specified on the command line ***override*** those specified using the configuration tool.

The following startup message is displayed:

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

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

```
CCL6502I: [ Initial ConnectionManagers = 10, Maximum ConnectionManagers = 100,
CCL6502I: Initial Workers = 10, Maximum Workers = 100, Port = 2345 ]
```

To get help on the startup options, enter:

```
ctgstart ?
```

Stopping the Gateway

To stop the Gateway:

- If you have ***not*** started the Gateway with the -noinput parameter, you can stop the Gateway by typing the correct character and pressing the `Enter` key in the Gateway console session. The allowable characters may be localized for your country; the default characters allowed are the `Q` or `-` characters.

  You can determine which characters will stop the Gateway by simply pressing the `Enter` key in the Gateway console session. A message like the following is displayed:

  ```
  CCL6508I: Type Q or - to stop the CICS Transaction Gateway.
  ```
- If you ***have*** used the -noinput parameter, you must stop the Gateway process using some other method, for example:
  – Use the **kill** command

Chapter 7. CICS Transaction Gateway Terminal Servlet

This chapter describes how to use the Terminal Servlet to access CICS 3270 applications using a Web browser.

- "What is the CICS Transaction Gateway Terminal Servlet?" tells you what a servlet is, and in particular what the CICS Transaction Gateway Terminal Servlet does.
- "Installing and configuring the Terminal Servlet" on page 77 discusses how you install and configure the Terminal Servlet.
- "Using the Terminal Servlet" on page 83 describes what you can do with the Terminal Servlet, and how to invoke it from your Web pages.
- "CICS Transaction Gateway Terminal Servlet samples" on page 89 describes how to use the samples supplied with the Terminal Servlet.
- "Properties and parameters reference" on page 90 provides a complete reference to the servlet properties you can specify for the Terminal Servlet.
- "Migrating from the CICS Internet Gateway" on page 95 describes what you need to do to migrate from using the CICS Internet Gateway to using the Terminal Servlet.
- "CICS Transaction Server for OS/390 Web Interface" on page 98 describes how HTML templates generated for use by the CICS Web Interface can be used by the Terminal Servlet.

It is advisable to read through all of this chapter before configuring and using the Terminal Servlet.

What is the CICS Transaction Gateway Terminal Servlet?

Servlets are Java programs that run on a Web server machine, inside a Java-enabled Web server or servlet engine, for example, WebSphere Application Server (see Figure 6 on page 77). Java servlets have become popular as alternatives to Common Gateway Interface (CGI) programs—the Terminal Servlet is a replacement for the CICS Internet Gateway shipped with IBM CICS Clients Version 2, which was a CGI program.

A servlet can be loaded automatically when the Web server is started, or loaded when the first client request for the services of the servlet is made. Unlike CGI programs, servlets are persistent, that is, once they are loaded they stay running, waiting for additional client requests. This makes servlets faster than CGI programs, as no time is wasted in creating and destroying processes.

Being Java programs, servlets are portable across operating systems. And, because the servlet interface is a standard, they can be used with different Web servers on different platforms.

The Terminal Servlet allows you to access CICS 3270 applications using a Web browser. You create Web pages that invoke the Terminal Servlet to start a CICS transaction, display screen information sent from a CICS server, and send screens back to CICS.

The Terminal Servlet can:

- **Behave like a simple terminal emulator**.

  The CICS screen is displayed in the browser as part of an HTML form. When you press one of the buttons in the form, the form is submitted and data is sent to CICS.
- **Substitute data from a CICS screen into HTML template files**.

  An HTML template file is used to determine how CICS data appears in the browser. The Terminal Servlet replaces variables in the HTML template file with CICS screen information, and the resulting output is sent to the browser. For more information, see “Using variable substitution” on page 86.
- **Using server-side includes, incorporate variable information in a Web page, or drive a terminal emulator without user interaction**.

  Server-side includes are statements in an HTML file (.shtml) that are processed by the Web server before the Web page is returned to the browser. For example, you can use a server-side include to invoke the Terminal Servlet to display the value of fields in a CICS screen. For more information, see “Invoking the servlet with a server-side include” on page 85.
- **Map CICS screens to Web pages**.

  Using the page mapping properties of the Terminal Servlet, you can associate particular Web pages with particular CICS screens. For example, you can specify the Web page to be displayed when the terminal is idle, and no transaction is running. For more information, see “Page mapping properties” on page 92.

Like EPI programming, the Terminal Servlet allows you to create new front ends to existing CICS transactions. However, it also allows you to exploit the power, flexibility, and platform-independence of Java programming.

Note: The Terminal Servlet does not support DBCS fields in 3270 data streams.

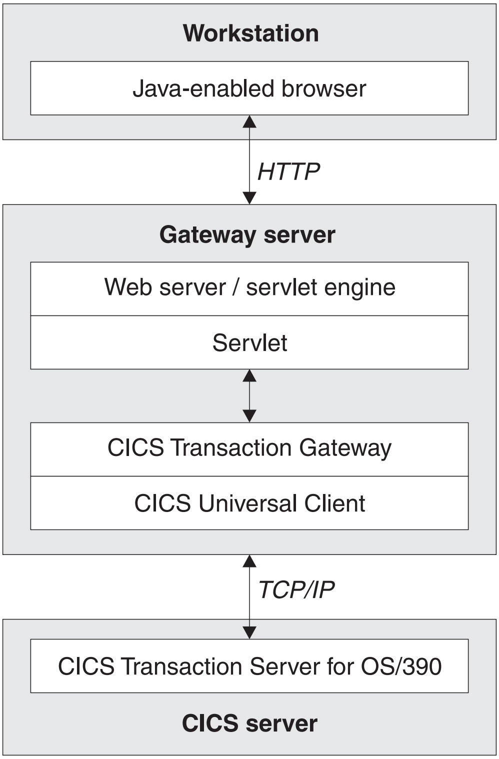


Figure 6. CICS Transaction Gateway using Terminal Servlet invoked by URL

Installing and configuring the Terminal Servlet

This section discusses how to install the Terminal Servlet on the Web server machine, and set the initialization parameters of the Terminal Servlet.

The procedure for installing and configuring the Terminal Servlet will vary according to the Web server. However, you will need to do at least the following:

- Configure the Web server's CLASSPATH and PATH settings
- Add the Terminal Servlet to the Web server's configuration
- Configure the servlet initialization parameters
- Consider other configuration options, such as security, logging, and whether the Web server can process server-side includes.

An example of installing and configuring the Terminal Servlet on a particular Web server is given in any configuration document that may be available, see "Sample configuration documents" on page 107. The principles described in that document should apply to other Web servers also.

Regardless of the Web server you use, you should refer to its documentation for guidance in installing and configuring servlets.

Configuring the Web server’s CLASSPATH and PATH settings

You must ensure that the two CICS Transaction Gateway jar files (**ctgclient.jar** and **ctgserver.jar**) are in the Web server’s classpath. In the case of WebSphere Application Server, you do this in the Java Engine section of WebSphere’s Administration tool.

If you do not make the correct classpath settings, you may get a message to the effect that you cannot update the servlet, and that the class **com.ibm.ctg.servlet.TerminalServlet** cannot be loaded.

You must also ensure that the CICS Transaction Gateway bin subdirectory is in the Web server’s path.

Adding the Terminal Servlet to the Web server’s configuration

To add the Terminal Servlet to the Web server’s configuration, you must provide a Servlet Name and associate this servlet instance with the Terminal Servlet class, which is **com.ibm.ctg.servlet.TerminalServlet**. In the case of WebSphere Application Server, you do this in the Servlet Configuration section of WebSphere’s Administration tool.

You can configure more than one instance of the Terminal Servlet. Each of them would have a different Servlet Name, but would refer to the same Servlet Class. In fact, you must create a different Terminal Servlet instance for each CICS server that you want to connect to. Each instance of the Terminal Servlet may use different initialization parameters, and this can be useful when configuring for different CICS servers.

Configuring the servlet initialization parameters

After adding the Terminal Servlet to the Web server’s configuration, you must set up the initialization parameters for the Terminal Servlet. In the case of WebSphere Application Server, you do this in the Servlet Configuration section of WebSphere’s Administration tool.

The following sections give some insight into how the Terminal Servlet works, which will help you in setting appropriate servlet initialization parameters.

The full set of initialization parameters are listed in “Properties and parameters reference” on page 90. However, to start with, you should consider setting some of the following parameters:

Table 3. Servlet initialization parameters

Parameter name	Notes
servlet@propertyFile	The servlet can load configuration information from a properties file. A sample servlet.properties file is supplied, in the subdirectory /samples/Java/com/ibm/ctg/servlet. To use a properties file, set this parameter to the full path name of the properties file you want the Terminal Servlet to load. If you specify a properties file then you may want to use this file for all other servlet parameters. In this case, **servlet@propertyFile** is the only parameter that you need to set.
servlet@extendedLogging	The servlet writes logging information, for example, error messages, to the standard servlet log. Set this property to `true` to write more information to the log file. You may find this helpful when first configuring the servlet, but it should be turned off once the servlet is set up correctly.
pool@maxTerminals	Set this property to the maximum allowable number of connections to the CICS server. You should ensure that the CICS server has sufficient free tasks to handle this number of concurrent connections. You should also ensure that the CICS Client **Maximum requests** configuration setting is at least as large as this property.
pool@serverName	Each instance of the Terminal Servlet can connect to only one CICS server. This gives you more control of resource usage and security. Set this property to the name of the required CICS server, as defined by the appropriate **Server name** setting in your configuration. If this property is not set, the default CICS server is used.
pool@gatewayUrl	Set this property if you want the servlet to connect over the network to the CICS Transaction Gateway.

Apart from the name of the property file, all the servlet properties can either be loaded from a properties file or set as servlet initialization parameters. Initialization parameters override properties file entries.

The servlet property names are not case sensitive.

Terminal pooling

The Terminal Servlet uses a pool of terminals, and the servlet configuration properties allow you to control the behavior of the terminal pool. The default behavior is that a new terminal is connected to CICS whenever a user requires one, up to the **pool@maxTerminals** limit. When the user has finished with the terminal, it is disconnected.

If the **pool@minTerminals** property is set, this number of terminals is connected to CICS when the servlet starts, and up to that number may be connected to CICS but not in use at any time. This allows new users to be allocated a terminal immediately, without waiting for a connection.

Setting the **pool@reusingTerminals** property allows terminals to be reused, that is, when a user session ends, the terminal may be allocated to a new user instead of being disconnected. This property has no effect if **pool@minTerminals** is 0. Any transaction running on the terminal is ended and the screen is cleared, before it is reused. However, if a user has signed on to CICS (using CESN for example), the terminal will still be signed on when it is reused. For this reason, do not set this property unless you are sure it is suitable for your environment.

The **pool@idleTimeout** property allows terminals that have been allocated to a user to be released if they have not been used for the specified timeout period. This property defaults to 0, that is, no timeout.

Using a display request with the values: **pool@connectedTerminals, pool@freeTerminals, and pool@terminalsInUse**, you can query the number of terminals that are connected, free, or in use for a particular instance of the Terminal Servlet.

Page mappings

Page mappings tell the Terminal Servlet what Web page to display for a particular CICS screen or terminal state.

The servlet’s choice of which page to display is based on the current page mapping settings, the current screen handler class, and the state of the terminal.

The Terminal Servlet first looks at the page mapping properties to see if there is a specific page set for the current state of the session, as follows:

Session State	**Page Mapping Property**
an error has occurred	**page@error**
the terminal is idle - no transaction is running	**page@idle**
the terminal has been disconnected	**page@disconnected**

Session State	Page Mapping Property
there is a screen handler for the current screen	Page mapping for the class name of the screen handler

If no specific page is found, the page identified by the page mapping property **page@default** is used. If this is not set, an error message is displayed instead.

The value of a page mapping property can be set in two ways:

1. A URL, for example:

   ```
   http://webserver/index.html
   ```

2. A template file to be loaded for variable substitution. This must start with ″file://″, for example:

   ```
   file://d:/pages/template.html
   ```

   If you set the **servlet@templateDir** property, template filenames can be specified relative to the template directory.

Example settings for page mapping properties are as follows:

```
page@error=http://server/errors.html
page@idle=http://server/idle.html
page@disconnected=/ctglab/html/servlet/epissam.html
page@default=file://epissam1.html
```

To set a page mapping for a particular screen, you must create a Screen Handler bean that can recognize that screen. Then, you must set the **pool@handlerPath** property to ensure that the Screen Handler bean is loaded by the Terminal Servlet. If, for example, you have created a Screen Handler bean called MAP1ScreenHandler in a package called testmaps, then the page mapping property for that screen handler is:

```
page@testmaps.MAP1ScreenHandler.
```

The name of the class is case sensitive. As for the standard page mappings, you set the property to the name of the Web page to display, for example:

```
page@testmaps.MAP1ScreenHandler=http://server/test.html
```

A page mapping can also be set in a request parameter to the Terminal Servlet, in which case it applies only to that request and overrides any page mappings associated with the instance of the servlet being used. You can query the current setting of a page mapping with an appropriate request (see “Displayable properties” on page 94).

Screen Handler beans and terminal disconnection

For a terminal to be disconnected successfully, it must exit from running transactions. The Terminal Servlet’s default behavior for achieving this is to send the AID (Attention identifier) PF3 key to CICS. However, if you will be running transactions for which this will not work, you should create one or more Screen Handler beans that know how to exit the relevant screens.

You can generate Screen Handler beans automatically from BMS maps using the BMS Map Conversion Utility (BMSMapConvert) supplied with the CICS Transaction Gateway, or you can write your own classes. For information about Screen handlers and how to create them, see the *CICS Transaction Gateway Programming* book.

To load screen handlers into the servlet, set the **pool@handlerPath** property. The handler path can include .jar files, .zip files and directories, although sub-directories will not be searched. For example, in the servlet properties file you might add:

```
pool@handlerPath=d:/handlers/handlers.jar;d:/handlers/test
```

to load screen handler classes from a .jar file and from a directory. See the sample properties file for more information on setting this property.

You can also specify a default screen handler (using the property **pool@defaultHandler**), which is used to exit from a screen if no other screen handler recognizes it. The sample default screen handler is **com.ibm.ctg.epi.DefaultScreenHandler**, which sends the AID PF3 to CICS. The source for this class can be found in the directory samples/java/com/ibm/ctg/epi. Any screen handler bean that can be loaded from the classpath or from the handler path can be used as the default screen handler.

The servlet will not keep trying to exit a running transaction indefinitely. The property **pool@exitRetryLimit** controls how many attempts it makes to exit the transaction. The default value of this property is 10, but you should ensure it is large enough for your environment. If the servlet finds there is still a transaction running on the terminal after this many attempts to exit, the terminal is assumed to be ‘dead’ and it will not be disconnected or allocated to a user. The only way to discard dead terminals is to unload the servlet, stop and restart the CICS Transaction Gateway, and load the servlet again.

If you have configured the servlet properly and provided Screen handler beans, you should not have any dead terminals.

Considering other configuration options

Other configuration options that you should consider are:

- **Security**

  You may need to control access to the Terminal Servlet.

- **Logging**

  The Terminal Servlet writes log information to the standard servlet log. You can set logging on or off.
- **Session tracking**

  You should configure the Terminal Servlet to use session tracking.
- **Server-side includes**

  You may wish to configure your Web server to process server-side includes.

Note: IBM WebSphere Application Server, Version 1.01, does not fully support input or output of DBCS data by servlets, and other servlet engines may have the same restriction. In this situation, the Terminal Servlet displays DBCS characters as ???? (question marks).

Loading the Terminal Servlet

After you have configured the Terminal Servlet and saved the settings, you must load it. You can choose whether to load it immediately, or each time your Web server starts up.

If you have not configured the Terminal Servlet properly, error messages are displayed when Web server attempts to load the Terminal Servlet. For more information refer to the Terminal Servlet section in the *CICS Transaction Gateway Messages* book.

Using the Terminal Servlet

This section describes how to use the Terminal Servlet. It describes the different ways in which the Terminal Servlet can be invoked, and also how to develop the Web pages that use the Terminal Servlet.

Connecting to CICS and starting a transaction

An instance of the Terminal Servlet can only connect to one CICS server. For each CICS server that you want to allow access to, you will have created an instance of the servlet. You might also have created instances of the servlet that have different initialization parameters.

To start a transaction on the CICS server, you must invoke the Terminal Servlet with a suitable request.

Invoking the Terminal Servlet

There are three ways to invoke the Terminal Servlet:

- by URL
- with an HTML form

- with a server-side include.

In each case, you need to know the name of the servlet instance that you want to use, and you pass the servlet one or more request parameters that tell it what action to take. The parameter names and values are the same in all cases, but the way you specify them is different.

To start a transaction on the CICS server, the parameters you must provide are:

Parameter Name	**Parameter Value**
request	send
transaction	the transaction ID of the transaction you want started. Note however that ATI transactions cannot be started for the Terminal Servlet.

The servlet will allocate a terminal to the user, if it is required.

The full set of request parameters are listed in “Request parameters” on page 93.

Invoking the servlet by URL

To invoke a servlet instance called TerminalServlet, link to the URL:

```
http://your_webserver/servlet/TerminalServlet
```

where *your_webserver* is the hostname of your Web server.

You can encode request parameters in the URL in the form of a query string, for example:

```
http://your_webserver/servlet/TerminalServlet?request=send&transaction=CECI
```

You should use this method of invoking the servlet when you do not need the user to enter any information.

Invoking the servlet with an HTML form

With this method, you create an HTML form in a Web page. For example:

```
<FORM METHOD="GET" ACTION="/servlet/TerminalServlet">
(Tags to place text entry areas, buttons, and other prompts go here)
</FORM>
```

is a form that would invoke the servlet called TerminalServlet. The METHOD can be GET or POST. The various elements of the form have names and values, and these are sent to the servlet when the form is submitted. If you know you want to set a particular request parameter, add it as a hidden item in the form, for example: <input type=″hidden″ name=″request″ value=″send″>. Hidden inputs are always sent to the servlet. In general, the name of a form element is the name of a request parameter, and its value is the value of the request parameter.

This is the only way to send user-entered information to the servlet.

The supplied sample, epissam3.html, illustrates the HTML form method, see “CICS Transaction Gateway Terminal Servlet samples” on page 89.

Invoking the servlet with a server-side include

A server-side include is processed by the Web server before the Web page is sent to the user. The servlet is invoked, and any output it generates is included in the Web page in place of the server-side include tags. The user only sees the servlet output in the Web page.

In the HTML source, a server-side include of the servlet looks like this:

```
<SERVLET NAME="TerminalServlet" >
<PARAM NAME="request" VALUE="send">
<PARAM NAME="transaction" VALUE="CECI">
<PARAM NAME="display" VALUE="none">
</SERVLET>
```

A server-side include can be used to add variable information to a Web page, or to drive a terminal to perform some action whenever a Web page is displayed, without any user involvement.

You can add as many server-side includes to a Web page as you wish, but **note**:

- Server-side includes in one page may be processed in parallel. You can therefore not be sure in which order they are performed, although usually this is the order in which they occur in the source.
- The servlet stores session information that allows it to access the terminal allocated to a user. If the user session is invoked by a server-side include, subsequent server-side includes in the same page will not be able to access the session information. Once the page has been sent to the user, the session is established, and a page with multiple server-side includes will work correctly.

The supplied sample epissam2.shtml, illustrates the server-side include method, see “CICS Transaction Gateway Terminal Servlet samples” on page 89.

What happens next?

When the servlet has been invoked from a form or by linking to a URL, some output must be sent back to the browser. To decide what should be displayed to the user, the servlet uses the page mapping properties that you have set up, see “Page mappings” on page 80.

Displaying screens and fields

There are two ways to include CICS screen information in a Web page:

- server-side includes
- *variable substitution*.

You can mix the two approaches if you want to.

Using server-side includes

Your Web server must be configured to process server-side includes. You create SHTML pages that contain server-side includes that invoke the servlet. The output from the servlet is inserted into the page in place of the server-side include.

For example, to display the screen (as part of an HTML form):

```
<SERVLET NAME="TerminalServlet" >
<PARAM NAME="display" VALUE="screen">
</SERVLET>
```

or to display the contents of the 22nd field on the screen:

```
<SERVLET NAME="TerminalServlet" >
<PARAM NAME="display" VALUE="22">
</SERVLET>
```

Using variable substitution

You create HTML template files that the servlet will load. The servlet replaces variables in the file with CICS screen information, and the resulting output is sent to the browser. You use page mapping properties to tell the servlet what template file to load (see “Page mappings” on page 80).

For example, a template file for the CESN signon screen might look like:

```
<html>
<head>
<title>CICS Signon</title>
</head>
<body>
<h1>CICS Signon</h1>
<form method="POST" action="/servlet/TermServlet">
```

```
<input type="hidden" name="request" value="send">
<pre>
 Userid: <input type="text" name="USERID" value="&USERID;" size="8" >
 Password: <input type="text" name="PASSWORD" value="&PASSWORD;" size="8" >
</pre>
<input type="submit" name="DFH_ENTER" value="Sign on">
<input type="submit" name="DFH_PF1" value="Help">
<input type="submit" name="DFH_PF3" value="Exit">
</form>
</body></html>
```

where the variables are &USERID; and &PASSWORD;. You can identify fields by number - to use names for fields, as in this example, you must have a Screen Handler bean which can recognize the screen and set fields by name.

Page mapping property values can be the names of HTML template files or any URL. So that the servlet knows that a page mapping property identifies an HTML template, you should set it to the value file://*template_filename*, for example: `file://d:/html/templates/test.html`.

What can be displayed?

Any piece of information that can be displayed by a server-side include can also be used for variable substitution, and vice-versa. The variable name used in a template is the same as the value of the **display** parameter in the server-side include.

Generally, you can display:

- The screen, as part of an HTML form
- A field - identified by number, or by name, if you have a Screen Handler bean for it
- Servlet configuration settings
- The number of terminals that are currently connected, in use, and free.

The full set of properties that can be displayed is listed in “Properties and parameters reference” on page 90.

Sending the screen back to CICS

To send the screen back to CICS you must invoke the servlet with the **request** parameter set to ″send″. You do not need to provide a **transaction** parameter - it is ignored if a transaction is already running. To update the screen with new information before sending it back to CICS, you specify further request parameters that identify the fields (by name or number) and what they should be set to.

For example, the following server-side include:

```
<SERVLET NAME="TerminalServlet" >
<PARAM NAME="request" VALUE="send">
<PARAM NAME="USERID" VALUE="sysad">
<PARAM NAME="PASSWORD" VALUE="sysad">
<PARAM NAME="display" VALUE="none">
</SERVLET>
```

would set the fields named USERID and PASSWORD to the values given, and then send the screen to CICS. In an HTML form, you might use the form element <input type=″text″ name=″USERID″ value=″″ size=″8″>. When the user submits the form, the request parameter USERID is set to the value that they entered.

Setting the AID

The default AID is Enter.

To set the AID from an HTML form, include a Submit button, specified like this:

```
<input type="submit" name="DFH_PF3" value="Exit" >.
```

If the user presses the button, the servlet is sent the request parameter ″DFH_PF3″ with the value ″Exit″ (which the servlet ignores). The AID is set to PF3. In a server-side include, you could set the AID like this:

```
<SERVLET NAME="TerminalServlet" >
<PARAM NAME="request" VALUE="send">
<PARAM NAME="DFH_PF3" VALUE="ignored">
<PARAM NAME="display" VALUE="none">
</SERVLET>
```

The full set of request parameters is listed in see “Properties and parameters reference” on page 90.

Disconnecting

When the servlet has allocated a terminal to a user, it is retained until:

- The servlet is invoked with the parameter ″request″ set to ″disconnect″
- The user’s session on the Web server expires
- If the **pool@idleTimeout** property has been set, the terminal is automatically disconnected if it is not accessed for the specified period of time.
- The CICS Client is stopped
- The connection to the CICS server is lost - or the server goes down
- The servlet is unloaded by the Web server.

You should try to explicitly release terminals when they are no longer required.

Note that when a user has established a session with the servlet and been allocated a terminal, any instance of the servlet to which the user has access can continue the session. The same terminal is used until the user disconnects or the session is ended for some other reason. The page mappings and other properties used for any particular request are those set for the instance of the servlet that is processing the request.

Properties that can be set as request parameters, for example, page mappings, override the settings for the servlet, but usually only apply to the current request, and do not affect requests by other users or subsequent requests by the same user.

CICS Transaction Gateway Terminal Servlet samples

This section describes how to use the samples provided with the Terminal Servlet.

Setting up the samples

The samples assume that you have configured the Terminal Servlet on your Web server with the instance name ‘TerminalServlet’. If you have used a different instance name, you will need to edit each of the sample HTML files and change ‘TerminalServlet’ to the correct name.

Copy all the sample HTML files to a suitable directory on your Web server.

Edit the sample properties file servlet.properties and set the following properties, using the instructions in the file:

1. **page@disconnected**
2. **page@default**
3. **page@demo.MAPINQ1ScreenHandler**
4. **pool@handlerPath**

Set the servlet initialization parameter **servlet@propertyFile** to the filename of the file servlet.properties.

To pick up the changes, unload and load the servlet.

Using the Terminal Servlet samples

These samples show you some of the ways you can use the CICS Transaction Gateway Terminal Servlet.

Basic Terminal Emulation

This sample shows you how to use the servlet as a replacement for a CICS terminal. It uses a server-side include to display the CICS screen as part of an HTML form. If you do not want to use server-side includes, change the value of the **page@default** property in the sample properties file to **file://[filename of epissam1.shtml]**.

The next two samples use the transaction EPIC, which is a sample transaction supplied with the CICS Transaction Gateway. You can find the source code for the program epiinq.ccs and for the map epiinq.bms in the subdirectory samples/server. They also use the sample screen handler beans supplied in the file **sample.jar**. You must compile both the program and map on your CICS server system and define a transaction EPIC to run the program epiinq.

Server-Side Includes Sample (epissam2.shtml)

This sample shows you how to use server-side includes to display information from a CICS screen and to drive the terminal.

HTML Template Sample (epissam3.shtml)

This sample shows you how to use an HTML template file.

Administrator information sample (epissam4.shtml)

This sample shows you how to display information that might be useful to an administrator.

Properties and parameters reference

This section describes the Terminal Servlet properties

Servlet configuration properties

In general, these properties can be specified either in a properties file or as servlet initialization parameters when you configure the Web server. The names are not case sensitive. Values given as servlet initialization parameters override settings loaded from a properties file. Apart from the **servlet@debug** and **servlet@extendedLogging** properties, these settings cannot be changed while the servlet is running. However, using the request parameter **display**, you can query the current settings of a property, see "Displayable properties" on page 94.

servlet@propertyFile

The full path name of a properties file to load. This property can only usefully be set as a servlet initialization parameter.

servlet@debug

If set to true, turns tracing on for all instances of the servlet. You can turn tracing on or off while the servlet is running, and output is directed to standard output. Note that turning tracing on seriously impacts performance!

servlet@extendedLogging

If set to true, turns extended logging on. Output is directed to the servlet log.

servlet@allocateTimeout

The maximum time in milliseconds that the servlet will wait for a terminal to be allocated to a user. The default is 20000 (20 seconds).

servlet@templateDir

The name of a directory from which HTML template files can be loaded. Setting this property prevents users from viewing files that are not in the template directory or its subdirectories. If you do not set this property, the servlet can load any file on the Web server that it has access to.

servlet@coloring

If set to Y, colored fields are displayed in the color indicated in the 3270 datastream.

Due to the limitations of HTML, unprotected fields cannot be colored and always appear as black text on a white background.

If the property is not specified. the Terminal Servlet displays the screen with black text on a white background.

pool@maxTerminals

The maximum number of terminals that can be connected to CICS concurrently. The default value is 5.

pool@minTerminals

The number of terminals that should be kept connected to CICS when not in use. The default value is 0.

pool@serverName

The name of the CICS server to connect to. If this property is not set, the default CICS server, as defined in your configuration is used.

pool@deviceType

The terminal model definition to use. If this property is not set, the default is used.

pool@handlerPath

The path from which Screen Handler beans are loaded. You can include directories, .jar files, and .zip files, but subdirectories are not searched.

pool@defaultHandler

The screen handler to use to exit screens when no specific handler is found. If you set this to be one of the Screen Handlers loaded from the Screen Handler path, make sure that the handler path is set before the default handler.

pool@idleTimeout
The maximum time in milliseconds to allow a terminal to be left idle before it is automatically disconnected. The default value is 0, meaning no timeout.

pool@exitRetryLimit
The number of times that the servlet will try to exit a running transaction during terminal disconnection. This prevents the servlet looping, if a screen cannot be exited.

pool@reusingTerminals
If set to true, released terminals are kept connected for the next user. The default behavior is to disconnect terminals when a user session ends. You must also set **pool@minTerminals** for this property to have any effect.

pool@gatewayUrl
The URL of the CICS Transaction Gateway that the servlet should connect to. The default is to use a local Gateway, that is `local://`.

screen@cursoring
If set to true, JavaScript sets the screen cursor position. The default setting is false.

screen@stripBlanks
If set to true, the Web Browser formats the field text.

screen@stripEmptyRows
If set to true, screen rows that do not contain data are removed.

Page mapping properties

Page mappings associate Web pages with particular screens or terminal states. Page mappings can be specified in a properties file, or as servlet initialization parameters, when you configure the Web server. However, using the request parameter **display**, you can query the current settings of a page mapping, see “Displayable properties” on page 94.

Standard page mapping properties	Value
page@disconnected	The page to show when the terminal is disconnected.
page@error	The page to show when an error occurs
page@idle	The page to show when the terminal is idle, that is, no transaction is running

Standard page mapping properties	**Value**
page@default	The page if no other page is specified.
page@*screen*	The page to show for this screen.

Request parameters

These parameters are set only when a request is made. You can also specify any other property that can be set on a request, for example: **servlet@extendedLogging**.

Parameter name	**Parameter value**
request	Set to ″send″ to send a screen to CICS or ″disconnect″ to end a session If required, a new terminal session is started.
display	The name of a property that can be displayed, see “Displayable properties” on page 94.
translateText	Set to ″true″ to translate the characters <, > and & to &lt; &gt; and &amp; respectively. This is used when text is displayed.
transaction	The transaction ID of a transaction to be started. This is only used if the request is ″send″ and the terminal state is idle. ATI transactions cannot be started for the Terminal Servlet.
transactionData	Additional parameters to pass to a transaction. This is only used if the request is ″send″ and the terminal state is idle, and a transaction was specified.
DFH_CURSOR	The field where the screen cursor should be placed, as a field index (a number), or the name of the field, if a screen handler bean is available for the current screen. This parameter name must be in upper case.
DFH_ENTER, DFH_CLEAR, DFH_PA1 - 3, DFH_PF1 - 24	The value is ignored. Use one of these parameters to specify the AID key to be sent to CICS. This must be in upper case.
A field index	The text to be set in the field. Use to set the contents of a field when you know the field index.

Parameter name	Parameter value
usingSession	Set to false if no terminal session is required. If this parameter is set, any terminal used for this request is discarded after use.
matchOnIdle	Set to false by default. Normally the servlet only attempts to find a ScreenHandler for the current screen when the terminal state is ‘client’, that is, a conversational or pseudo-conversational transaction is expecting a response. Setting this property to true causes the servlet to check the current ScreenHandler every time the screen changes regardless of the state. You may want to do this if you have transactions that use BMS maps, but which are not conversational or pseudo-conversational. If you have a large number of ScreenHandlers, the servlet may run more slowly when this property is set to true. For this reason, the default value is false.

In addition, if a Screen Handler bean is available for the current screen, you can set any properties on that bean by specifying them in the request. For beans generated by BMSMapConvert, the field names defined in the BMS map are properties that can be set to the required text in the field.

Displayable properties

The following table shows the values that can be specified for the ″display″ request parameter or used for variable substitution, see “Using variable substitution” on page 86.

Display values	Displays
none	Nothing. Use to suppress output from a server-side include.
screen	The screen - as part of an HTML form.
errorMessage	If an error has occurred, the error message.
errorCause	If an error has occurred, further information about the error.
DFH_CURSOR	The name of the field where the cursor is, if it can be determined, otherwise the field index.
a field index	The field text

Display values	Displays
a screen handler property name	The value of the property if it can be determined.
any of the servlet configuration properties listed above	The value of the property for the current servlet instance.
any page mapping property	The value of the property for the current servlet instance.
any request parameter, except for ″request″ and ″display″	The value of the property for the current request.
pool@connectedTerminals	The current number of connected terminals, for this servlet instance.
pool@freeTerminals	The current number of connected terminals that are not in use, for this servlet instance.
pool@terminalsInUse	The current number of terminals allocated to users, for this servlet instance.

Migrating from the CICS Internet Gateway

This section describes how to migrate from using the CICS Internet Gateway to using the Terminal Servlet.

Before you perform migration you should be familiar with the concepts of the Terminal Servlet as described in this chapter.

The CICS Internet Gateway reads settings from a configuration file, which is called **cigd.env**.

CICS Internet Gateway default settings

CICS Internet Gateway	Terminal Servlet equivalent
Cursor	The Cursor key equivalent. There is no equivalent setting for the Terminal Servlet. Instead, you can use JavaScript to set the position of the cursor. To allow this, set the servlet property **screen@cursoring** to true.
Info	Filename for the information file for the server (for AIX and Windows NT). There is no equivalent setting for the Terminal Servlet.
Error	Filename for the error file for the server. There is no equivalent setting for the Terminal Servlet.

Trace

Filename for the trace file for the server. There is no equivalent setting for the Terminal Servlet. Servlets write logging information to the servlet log provided by the Web server. See your Web server documentation to find out where this file is. If trace is turned on it is written to the standard output. Depending on your Web server, this may be directed to a file.

MaxUsers

The maximum number of users. The equivalent servlet property is **pool@maxTerminals.**

TimeOutCICS

The “Wait for CICS ” timeout. The servlet property **servlet@allocateTimeout** limits the time that the servlet will wait for a terminal to be connected to CICS. There is no limit on the time the servlet will wait for CICS to respond in other situations.

TimeOutGateway

The “Wait for Gateway” timeout. There is no equivalent setting for the Terminal Servlet.

TimeOutInternet

The “Wait for Internet” timeout. The servlet property **pool@idleTimeout** limits the time that the servlet will let a terminal be unused before it is disconnected.

CICS Internet Gateway override settings

CICS Internet Gateway — **Terminal Servlet equivalent**

AutoExit

Normally, the Terminal Servlet does not disconnect the terminal when a transaction ends. If this is the behavior you want, set the page mapping property **page@idle** to the URL /servlet/TerminalServlet?request=disconnect, replacing TerminalServlet with the name of an instance of the Terminal Servlet on your Web server.

ExitAid

The Exit key equivalent. The servlet uses Screen Handler beans to exit running transactions. You can generate and customize Screen Handler beans for specific CICS screens, and you can create a default Screen Handler that knows how to exit a range of transactions. The supplied sample DefaultScreenHandler exits using the AID PF3.

ExitPage
The URL for the HTML exit page. Use the page mapping property **page@disconnected** to specify a page to be displayed when the terminal is disconnected.

ImbedHTML
Allows Imbedded HTML. The servlets default behavior is to write out the contents of CICS screens as is, that is, text taken from the CICS screen is assumed to be correct HTML. If the characters <, >, or & are present on the CICS screen, this may cause problems. To translate the characters <,>, and & to &lt; &gt; and &amp; respectively, set the request parameter **translateText** to true. No translation is performed when sending data to CICS.

Header
The filename for the HTML header. There is no equivalent setting for the Terminal Servlet.

Trailer
Filename for the HTML trailer. There is no equivalent setting for the Terminal Servlet.

GraphicKeys
Display the keys as GIFs. There is no equivalent setting for the Terminal Servlet.

PAKeys
Display the PA keys. There is no equivalent setting for the Terminal Servlet.

PFKey24
Display extra row of PF keys. There is no equivalent setting for the Terminal Servlet. Output from the Terminal Servlet is inserted into a Web page that you provide. The sample file epissam1.shtml is provided as a starting point.

CICS Internet Gateway <FORM> actions

The CICS Internet Gateway is invoked by an HTML form. To invoke the Terminal Servlet instead, change the <FORM> action to /servlet/*servlet* where *servlet* is TerminalServlet or another name.

Query the current gateway settings
You can query the various servlet properties individually. See “Displayable properties” on page 94 for details.

List the available CICS systems
The servlet does not provide this function. Each Terminal Servlet instance can connect to only one CICS server. You should configure an instance of Terminal Servlet for each CICS server that you want to allow users to connect to.

Start a transaction

You must add a hidden <FORM> element setting the name ″request″ to the value ″send″, as in Table 4

Table 4. Terminal Servlet equivalents of <FORM> actions

CICS Internet Gateway <FORM> element name	Terminal Servlet equivalent
SystemName	None
TranName	transaction
TranData	transactionData
AutoExit	page@idle=/servlet/TerminalServlet?request=disconnect is equivalent to AutoExit=On
ExitAID	None
ExitPage	page@disconnected
ImbedHTML	translateText=true is equivalent to ImbedHTML=Off
Anything else	None

See “Properties and parameters reference” on page 90 for more information.

Administration functions

You can query the number of terminals that are in use and turn trace on and off. However, these functions cannot be restricted to administrators.

CICS Transaction Server for OS/390 Web Interface

The CICS Web Interface, included in CICS Transaction for OS/390 Version 1.2, is a collection of transactions and programs supporting direct access to CICS transaction processing services from Web browsers.

If you have generated HTML templates for use by the CICS Web Interface, they can also be used as HTML templates by the Terminal Servlet. You will have to make some changes to invoke the servlet instead of the CICS Web Interface, but the format is basically the same.

To use the template file:

1. Copy the template file to a directory on the Web server.
2. Edit it to change the <FORM> action to /servlet/TerminalServlet and add a hidden <FORM> element with name ″request″ and the value ″send″.
3. Generate a Map class and Screen Handler bean for the BMS map that the HTML template refers to.

4. Compile the generated Java source files.
5. Copy the compiled class files to a suitable directory on the Web server.
6. Set the servlet property **pool@handlerPath** so that it will load the new classes.
7. Set a page mapping property to associate the new Screen Handler bean with the HTML template file.

Chapter 8. Problem determination and problem solving

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?
- Is your CLASSPATH path set correctly? (See “Configuring your programming environment for CICS Transaction Gateway” on page 29.)
- Is the CICS Universal Client environment set correctly?
- Are there any messages explaining the failure?
- Can the failure be reproduced?

What to do next

If you think the problem is in the CICS Transaction Gateway, 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 in the CICS Universal Client or elsewhere in the network, you should follow the problem determination procedures provided with those other products. CICS Universal Clients problem determination is described in the *CICS Clients Universal Administration* book for your platform; for CICS Servers, see the *Problem Determination* book for the relevant server.

Using trace

The CICS Transaction Gateway trace records information passing between the browser and the Gateway.

Problem determination

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. For more information on starting CICS Transaction Gateway, see “Starting the Gateway” on page 71.

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

Program support

To obtain program support for the CICS Transaction Gateway you need to report the problem against the **CICS Universal Client** version you are using.

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 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, see the *CICS Transaction Gateway Messages* book.

Problems with the JDK AppletViewer

When using AppletViewer to run any of the CICS Transaction Gateway sample applets from your local file system, the message

```
CCL6664E Unable to load relevant class to support the protocol protocol
```

may be displayed.

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

Terminal Servlet problems

A servlet log is provided by the servlet engine that runs the Terminal Servlet, and all instances of the servlet write to the same log. For WebSphere the servlet log is:

```
\Websphere\Appserver\logs\servlet\servletservice\event_log
```

The Terminal Servlet writes to the log when it is started or stopped, and whenever an error occurs. When the servlet engine initializes or destroys an instance of the servlet, it is also recorded in the servlet log.

Using the **servlet@extendedLogging** property, you can control whether extended logging is performed. If extended logging is turned on, additional information is written to the servlet log, that is, the servlet writes to the log every time it receives a request. This can provide further valuable information, but you should turn extended logging off when not required, as it slows down the servlet.

Using the **servlet@debug** property, you can control whether tracing is performed for the Terminal Servlet. If tracing is turned on, full Gateway trace is turned on for every servlet instance. The destination of the trace output depends on the servlet engine. For WebSphere, you must enable trace by editing the file:

```
\Websphere\AppServer\properties\bootstrap.properties
```

and setting the following:

```
java.debug=true
```

and trace output will be written to the \Websphere\AppServer\logs\jvm_stdout.log and jvm_stderr.log files. You should turn this tracing off when not required, as it slows down the servlet.

You can turn both extended logging and tracing on while the Terminal Servlet is running. For more information on Terminal Servlet properties see “Properties and parameters reference” on page 90.

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 Universal Clients books” on page 106
- “CICS Family publications” on page 106
- “Book filenames” on page 107
- “Sample configuration documents” on page 107
- “Other publications” on page 108
- “Viewing the online documentation” on page 108

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*

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

Table 5. 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
Anynet
CICS
eNetwork
IBM
MQSeries
MVS
MVS/ESA
OS/2
OS/390
OS/400
RETAIN
TXSeries
VisualAge
VSE/ESA
VTAM
WebSphere

Lotus Domino, Lotus Notes, and Domino Go Webserver are trademarks of Lotus Development Corporation.

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

Program Number: 5648-B43

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

SC34-5591-00