WebSphere software

IBM

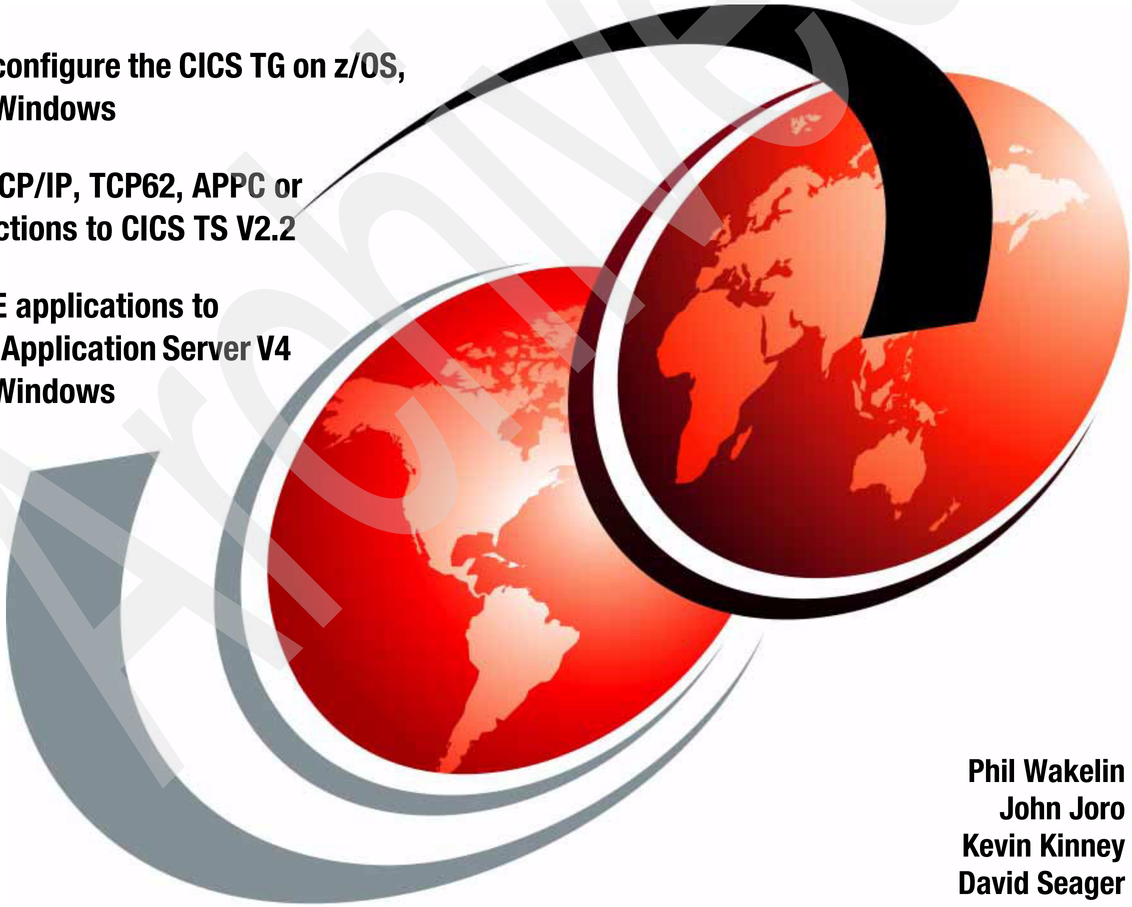
CICS Transaction Gateway V5

The WebSphere Connector for CICS

Install and configure the CICS TG on z/OS, Linux, and Windows

Configure TCP/IP, TCP62, APPC or EXCI connections to CICS TS V2.2

Deploy J2EE applications to WebSphere Application Server V4 on z/OS or Windows



Phil Wakelin
John Joro
Kevin Kinney
David Seager

Redbooks

ibm.com/redbooks

International Technical Support Organization

CICS Transaction Gateway V5
The WebSphere Connector for CICS

August 2002

SG24-6133-01

Note: Before using this information and the product it supports, read the information in "Notices" on page ix.

Second Edition (August 2002)

This document updated on June 26, 2009.

This edition applies to CICS Transaction Gateway V5.0 and CICS Transaction Server for z/OS V2.2 for use on the z/OS and Windows operating systems.

Contents

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.

The following paragraph does not apply to 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 publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication 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.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

Preface

The CICS® Transaction Gateway (CICS TG) is widely used to provide access to CICS COMMAREA-based programs and 3270 transactions from Java environments. This IBM® Redbook shows you how to build a robust CICS TG configuration for a variety of different configurations.

First we introduce the facilities of the CICS TG, followed by step-by-step explanations of how to use the different protocols (TCP/IP, TCP62, APPC, and EXCI) used for communication with a CICS TS V2.2 region on z/OS®, and how to secure your CICS region when receiving External Call Interface (ECI) or External Presentation Interface (EPI) requests.

Next, we provide details on how to configure the CICS TG V5 on either z/OS or Linux to connect a Java client application to a CICS region. The use of the Secure Sockets Layer (SSL) protocol to encrypt the communication from the Java application to the CICS TG is included in these scenarios.

Finally, we offer two scenarios to illustrate how to configure WebSphere® Application Server V4 on the Windows or z/OS platforms, to use the supplied ECI resource adapter to allow J2EE applications to make ECI calls to CICS.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

.

The authors: Phil, Dave, Kevin, and John

Phil Wakelin is a Senior I/T Specialist at the International Technical Support Organization, San Jose Center, and has more than 12 years' experience working on most platforms and versions of CICS. He writes extensively and has taught IBM classes about many areas of CICS, specializing in CICS Web-enablement and client-server technology. Before joining the ITSO, Phil worked in the Installation Support Center, IBM UK as pre-sales support and in the CICS System Test department of the IBM Hursley Laboratory in the UK.

John Joro is an I/T Specialist at IBM Global Services in Finland. He has 15 years of experience working on OS/390® with CICS and CICSPlex® SM.

Kevin Kinney is a senior systems programmer at Unisure in Cincinnati, Ohio. He has 15 years of experience in mainframe technologies and has worked at Unisure for six years. His areas of expertise include everything OS/390.

David Seager is a Software Engineer in Hursley, UK. He has five years of experience working in the transaction processing field. He holds a degree in physics from Oxford University. His areas of expertise include Java, all things object-oriented, and many different platforms. He has written several DeveloperWorks articles on many different Internet technologies.

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

Geoff Sharman, Chris Smith, Jon Butts, IBM Hursley for supporting this project.

Richard Johnson, Robert Jerrom, Kate Robinson, Daniel McGinnes, Dave Kesley, Innes Reed, Simon Knights, David Radley, and Darren Beard, all of the IBM Hursley Lab, for explaining the workings of the CICS Transaction Gateway.

Nigel Williams of IBM Montpellier, Kevin Senior of IBM Italy, and Holger Wunderlich of ITSO Poughkeepsie, for insight into WebSphere security on z/OS.

Mark Endrei of ITSO Raleigh, Jim Grauel of IBM Raleigh, and Steve Wall of IBM Montpelier for their review comments.

Rich Conway, Bob Haimowitz, and Dave Bennin of the ITSO for providing excellent systems support.

Steve Day, Frances McKenna, and Shannan Read, who were the authors of the previous version of this IBM Redbook.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks® to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. QXXE Building 80-E2
  650 Harry Road
  San Jose, California 95120-6099

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-6133-01
for CICS Transaction Gateway V5
as created or updated on June 26, 2009.

August 2002, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described below.

New and changed information

The following topics have been added or modified in this Second Edition:

- Use of CICS Transaction Gateway V5.0
- Use of CICS TS V2.2, including ECI over TCP/IP support
- CICS ECI and EPI security scenarios
- Dynamic trace facility in CICS Transaction Gateway V5.0
- Use of Gateway daemon on Linux
- Use of multiple Gateway daemons on z/OS
- Use of transactional EXCI
- WebSphere Application Server V4.01 for z/OS and OS/390
- WebSphere Application Server V4.03 for Windows
- Use of JSSE for 128-bit Secure Sockets Layer support, in place of SSLight
- TCP62 dynamic LU name generation, including the tcp62locallu utility
- CallCICS servlet sample, replaced with CTGTesterECI and CTGTesterCCI samples

Note: In December 2002, this version was updated with minor technical corrections.

Part 1

Introduction

In this part, we give a broad overview of the CICS Transaction Gateway, the Java programming interfaces it offers, and the means of connecting the CICS Transaction Gateway to CICS regions on z/OS.

1

CICS Transaction Gateway

This chapter gives a broad overview of the CICS Transaction Gateway (CICS TG) and the features and functions it provides.

The following topics are discussed:

- CICS TG interfaces
- Gateway daemon
- Client daemon
- Configuration Tool
- Terminal Servlet

1.1 CICS TG: Infrastructure

The CICS Transaction Gateway (CICS TG) is a set of client and server software components that allow a Java application to invoke services in a CICS region. The Java application can be an applet, a servlet, an enterprise bean, or any other Java application (Figure 1-1).

The latest edition of the CICS TG is V5.00, and the currently supported platforms are: z/OS, OS/390, Linux for S/390®, AIX®, HP-UX, Sun Solaris, Windows NT, and Windows 2000.

The CICS TG is supported for use with CICS/ESA V4.1, CICS/VSE 2.3 and CICS Transaction Server (CICS TS) for VSE/ESA V1, but only if the CICS TG runs on a distributed platform. For use with CICS TS V1 for OS/390 or CICS TS V2 for z/OS, the CICS TG can run on z/OS, OS/390, or a distributed platform.

For product information on using CICS TG, refer to the *CICS Transaction Gateway Administration Guides*.

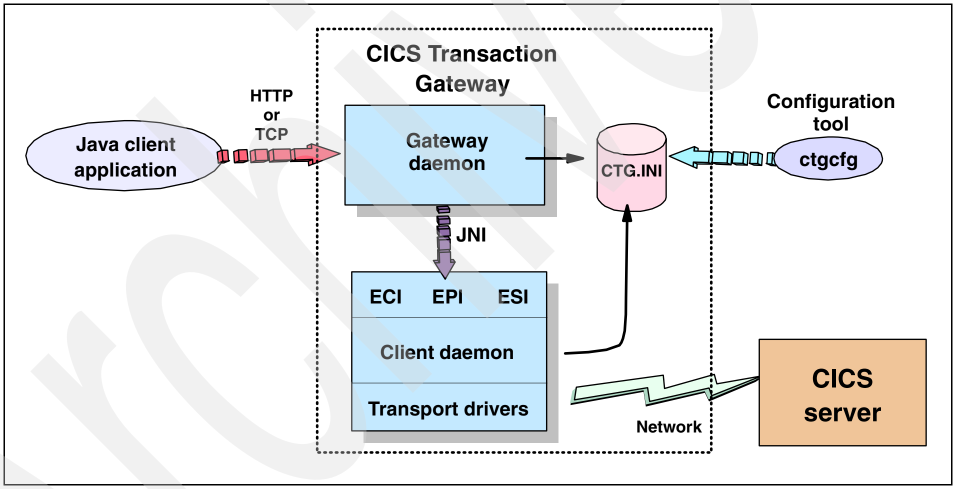


Figure 1-1 CICS Transaction Gateway components

The CICS Transaction Gateway consists of the following principal components:

- Gateway daemon
- Client daemon
- Configuration Tool (ctgcfg)
- Terminal Servlet
- Java class library

1.1.1 Client daemon

The CICS TG Client daemon is an integral part of the CICS TG on all distributed platforms. It provides the CICS client-server connectivity using the same technology as previously provided by the CICS Universal Client. On distributed platforms, connections to the following CICS servers are supported:

- APPC connections from Windows and AIX platforms to all CICS platforms
- TCP62 (LU 6.2 over IP) connections to CICS/ESA V4.1, CICS TS V1.2 and CICS TS V1.3 for OS/390, and CICS TS for z/OS V2
- TCP/IP connections to CICS TS for z/OS V2.2, CICS TS for VSE/ESA V1.1.1 the TXSeries CICS Servers (AIX, Sun Solaris, Windows NT, and HP-UX) and CICS OS/2 Transaction Server

For further details on supported platforms and required service levels, refer to the appropriate announcement letters available at the following URL:

http://www-4.ibm.com/software/ts/cics/announce/

The Client daemon runs in the background as the `cclclnt` process. Interaction with the Client daemon is provided by the `cicscli` executable, which can be used to start, stop and query the Client daemon, among other functions. The command **`cicscli -?`** lists the **`cicscli`** command options, as shown in Example 1-1.

On the Windows platform, the Client daemon can run as a Windows service. The ability to run as a service is provided by the `cclserv.exe` executable, which is registered as a service during CICS TG installation. The Client daemon can be started and stopped using the Windows Service Control Manager.

Example 1-1 Options for the cicscli command

```
C:\>cicscli -?
CCL8001I CICSCLI - CICS Client Control Program
CCL0002I (C) Copyright IBM Corporation 1994,2001.  All rights reserved.
CCL8002I Command options are:
CCL8003I   -S[=server]      - Start the client [and connect to a server]
CCL8004I   -X[=server]      - Close the client [or server connection]
CCL8005I   -I[=server]      - Abort the client [or server connection]
CCL8006I   -L               - List active server connections
CCL8007I   -D[=size]        - Enable service tracing [and set size limit]
CCL8008I   -O               - Disable all service tracing
CCL8009I   -C=server        - Specify the server for security changes
CCL8010I                      Must specify with one or both of /U and /P
CCL8011I   -U[=userid]      - Set the userid to be used with a server
CCL8012I   -P[=password]    - Set the password to be used with a server
CCL8013I   -N               - Suppress client pop-ups
CCL8014I   -E               - Activate client pop-ups
```

```
CCL8015I   -W              - Wait for confirmation before completing
CCL8016I   -Q              - Inhibit all output messages
CCL8017I   -V              - Display client version information
CCL8018I   -Y              - Perform a controlled restart of the client
CCL8019I   -J              - Perform an immediate restart of the client
CCL8020I   -M[=components] - Specify the components to trace
CCL8021E
CCL8023I CICSCLI performed no action
```

The **cicscli -l** command can be used to list which CICS servers are in use by the Client daemon, as shown in Example 1-2.

Example 1-2 Output from the cicscli -l command

```
C:\>cicscli -l
CCL8001I CICSCLI - CICS Client Control Program
CCL0002I (C) Copyright IBM Corporation 1994, 2002.  All rights reserved.
CCL8041I The CICS Client is using the following servers:
CCL8043I Server 'SC62PJA1' (using 'TCP62' to 'USIBMSC.SCSCPJA1') is unavailable
CCL8042I Server 'SCSCPJA1' (using 'TCPIP' to 'wtsc66oe.itso.ibm.com') is
available
CCL8044I Server 'SCSCPJA4' (using 'TCPIP' to 'wtsc66oe.itso.ibm.com') is
connecting
```

1.1.2 Gateway daemon

The Gateway daemon is a long-running process that functions as a server to network-attached Java client applications (such as applets or remote applications) by listening on a specified TCP/IP port. The CICS TG supports four different CICS TG *network protocols* (*TCP, SSL, HTTP,* or *HTTPS*), each of which requires a different CICS TG *protocol handler* to be configured to listen for requests (Figure 1-2).

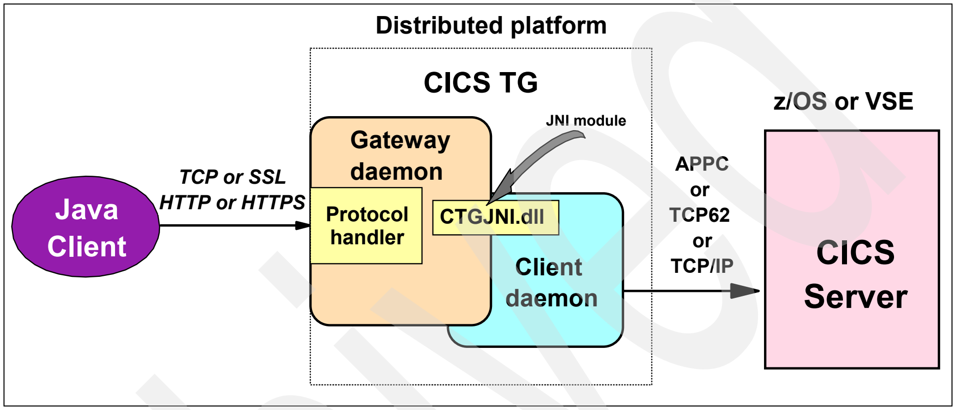


Figure 1-2 CICS Transaction Gateway: distributed platform

The structure of the Gateway daemon is slightly different on z/OS and on distributed platforms. On distributed platforms (including Linux for S/390), the CICS TG provides equivalent functions to those provided by the CICS Universal Client. There are three basic interfaces that are provided to Java client applications:

External Call Interface (ECI) A call interface to COMMAREA-based CICS applications

External Presentation Interface (EPI) An API to invoke 3270-based transactions

External Security Interface (ESI) An API that allows password expiration management (PEM) functions to be invoked in CICS, in order to verify and change user IDs and passwords

In V5 of the CICS TG, there is now also a new type of protocol handler called *TCPAdmin*, which is part of the dynamic trace facility. This allows a Java client to connect to the CICS TG and dynamically control the CICS TG trace settings. For further details on how we used this, refer to “Dynamic trace (gateway)” on page 174.

On z/OS, the External CICS Interface (EXCI) is used in place of the Client daemon, and provides access to COMMAREA-based CICS programs (Figure 1-3). Consequently, the EPI and ESI interfaces are not available with the CICS Transaction Gateway for z/OS. There are also a few differences between the ECI support on z/OS and the ECI support using distributed platforms.

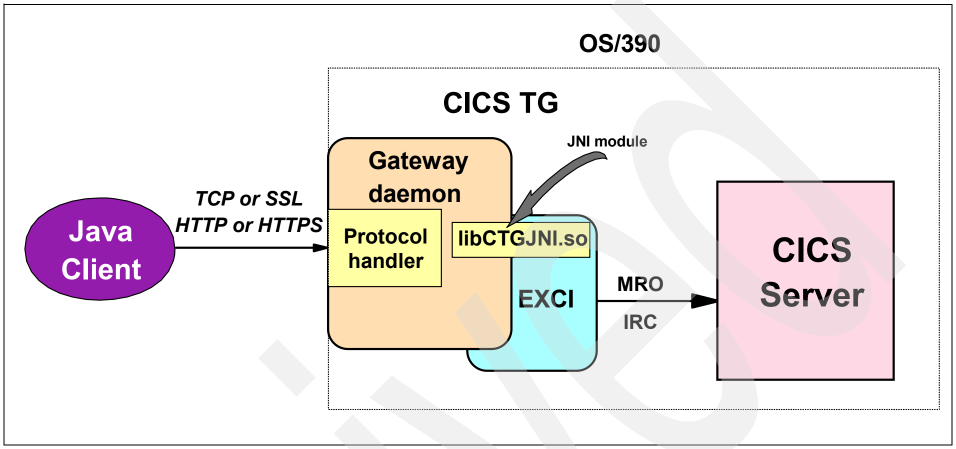


Figure 1-3 CICS Transaction Gateway: z/OS

The primary differences in the ECI support offered when the CICS TG is running on z/OS are as follows:

- When using asynchronous calls, specific reply solicitation calls are not supported.
- The user ID and password flowed on ECI requests are verified within the CICS TG with RACF®; afterwards the verified user ID is then flowed onto CICS.
- The `ECIRequest` method `listSystems()` does not return the list of usable servers, since any CICS region within the Parallel Sysplex® can be reached.

> **Note**: The Gateway daemon is not usually required when a Java application executes on the same machine as where the CICS TG is installed. In this situation, the CICS TG *local:* protocol can be used, which directly invokes the underlying transport mechanism using the Java Native Interface (JNI) module libCTGJNI.so.

1.1.3 Configuration Tool

The Configuration Tool (ctgcfg) is a Java-based graphical user interface (GUI) supplied by the CICS TG on all platforms. It is used to configure the Gateway daemon and Client daemon properties, which are stored in the CTG.INI file. In versions of the CICS TG prior to V3, the CICSCLI.INI and gateway.properties files were used to store the configuration parameters now stored in CTG.INI. Figure 1-4 illustrates the graphical user interface of the Configuration Tool.

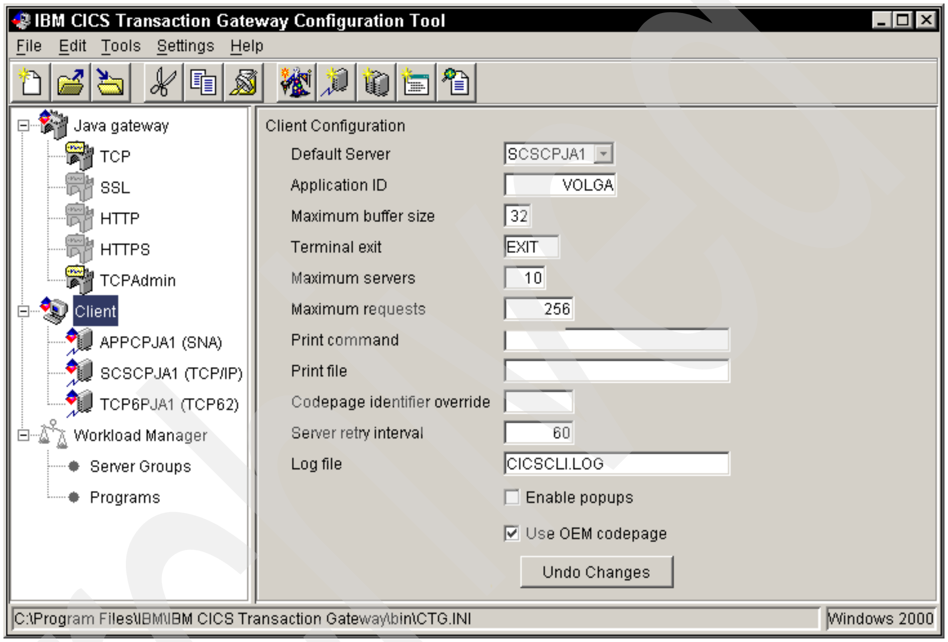


Figure 1-4 Configuration Tool

The Configuration Tool displays three main types of resources as follows:

- The different Java gateway protocol handlers (TCP, SSL, HTTP, HTTPS, and TCPAdmin). For details on how we configured the TCP, SSL and TCPAdmin protocol handlers, refer to the chapters in Part 3, “Gateway daemon scenarios”.
- The Client daemon, and the configured server connection to defined CICS regions. For details on how we configured TCP/IP, APPC and TCP62 connections to our CICS regions, refer to the chapters in Part 2, “Configuring CICS connections”.
- The Workload Manager and the Server Groups and Programs defined for workload balancing. For information on the CICS TG Workload Manager, refer to the redbook, *Workload Management for Web Access to CICS,* SG24-6118.

1.1.4 Terminal Servlet

The Terminal Servlet is a supplied Java servlet that allows you to use a Web browser as an emulator for a 3270 CICS application. It dynamically converts 3270 output into HTML for display on a Web browser, and is a non-programmatic solution for Web-enabling 3270 applications.

The Terminal Servlet is similar in function to the CICS-supplied 3270 Web bridge, in that it provides a turn-key solution to Web-enabling 3270-based applications. It also has the ability to use the same HTML templates as the 3270 Web bridge to display the output of CICS BMS maps as HTML forms. In addition, it provides a basic terminal emulation capability, and the ability to use *server-side includes* to display information from a CICS screen. The HTML template interface offered by the Terminal Servlet is shown in Figure 1-5.

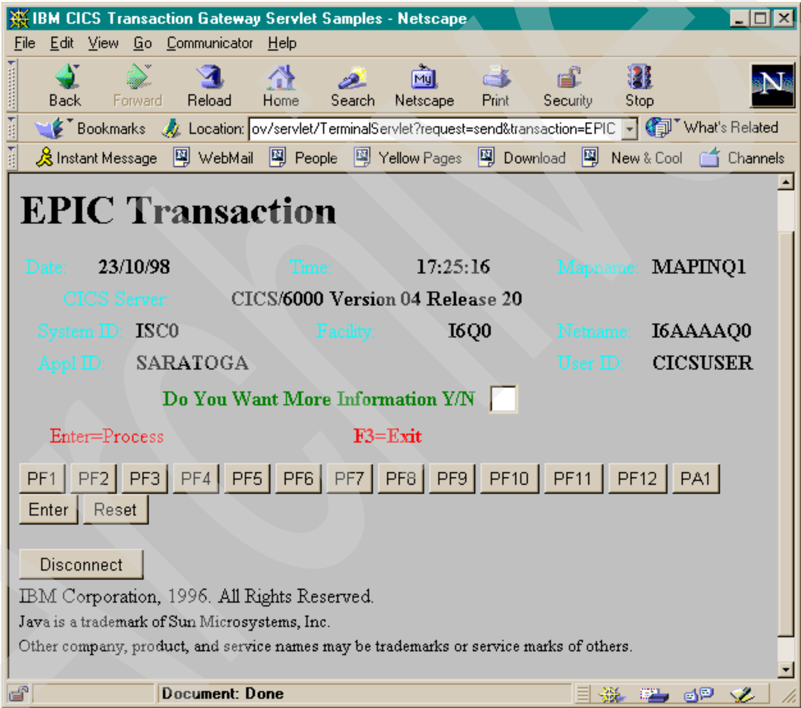


Figure 1-5 CICS TG Terminal Servlet

1.2 CICS TG: Interfaces

All the principal interfaces provided by the CICS TG fall into one of three categories, based on the function being invoked in CICS:

- External Call Interface (ECI)
- External Presentation Interface (EPI)
- External Security Interface (ESI)

For further details on programming with the CICS TG, refer to the redbook *Java Connectors for CICS*, SG24-6401.

1.2.1 External Call Interface

The ECI is used for calling COMMAREA-based CICS programs. The COMMAREA is the buffer that is used for passing the data between the client and the CICS server. CICS sees the client request as just another distributed program link (DPL) request (Figure 1-6).

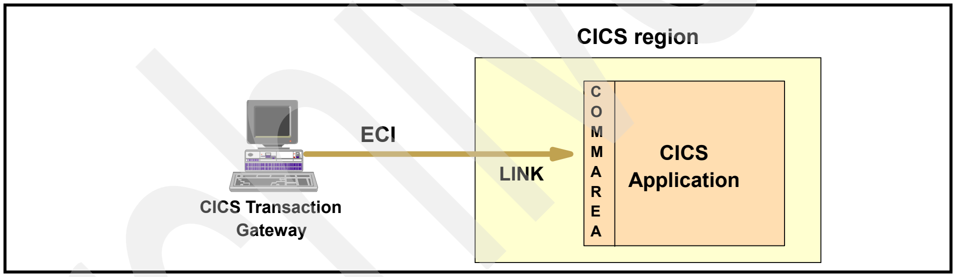


Figure 1-6 External Call Interface

An ECI request can be made in Java using one of three different interfaces:

- The `ECIRequest` class provided by the CICS TG base classes.

  This provides a simple procedural type interface to the ECI, and it is used for calling COMMAREA-based CICS applications.

- The Common Connector Framework (CCF) classes.

  These classes implement the CCF interfaces and programming model. The CCF is comprised of three core components: the CCF classes, the Enterprise Access Builder, and the Java Record Framework.

- The Common Client Interface (CCI) provided by the J2EE Connector Architecture.

  These classes define a standard architecture for connecting the Java 2 Platform Enterprise Edition (J2EE) platform to a heterogeneous EIS such as

CICS. Java applications interact with resource adapters using the Common Client Interface (CCI), which is largely based on the CCF, but it is a standard that is open to the entire Java community.

1.2.2 External Presentation Interface

The EPI is used for invoking 3270-based transactions. A terminal is installed in CICS, and CICS sees the request as running on a remote terminal controlled by the CICS TG. For further details on programming with the EPI, refer to Chapter 7, “EPI support classes”, in the redbook *Java Connectors for CICS*, SG24-6401.

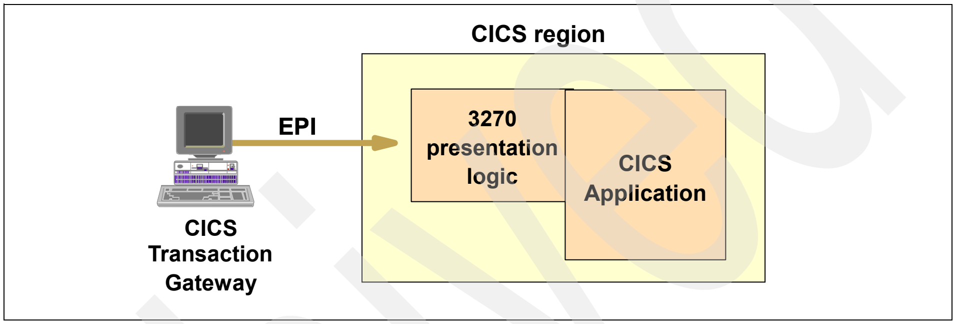


Figure 1-7 External Presentation Interface

An EPI request can be made in Java using one of five different interfaces:

- The `EPIRequest` class provided by the CICS TG base classes.

  This class provides a Java interface to the EPI, and is used for invoking 3270-based transactions. Due to its low-level nature, using it for developing EPI applications requires a strong knowledge of CICS and 3270 data streams.

- The EPI support classes, which provide high-level constructs for handling 3270 data streams.

  A wide range of classes is provided including `AID`, `FieldData`, `Screen`, `Terminal`, `Map and MapData`. These are used to represent the interface to a CICS 3270 terminal, and the resulting 3270 response.

- The EPI beans, which are based on the EPI support classes and JavaBean development environment.

  They allow you to create EPI applications in a visual development environment, using one of the visual application builder tools, such as VisualAge® for Java.

- The Common Connector Framework (CCF) classes.

  These classes implement the CCF interfaces and programming model. The CCF is comprised of three core components: the CCF classes, the Enterprise Access Builder, and the Java Record Framework.

- The Common Client Interface (CCI) provided by the J2EE Connector Architecture.

  These classes define a standard architecture for connecting the Java 2 Platform Enterprise Edition (J2EE) platform to a heterogeneous EIS such as CICS. Java applications interact with resource adapters using the Common Client Interface (CCI), which is largely based on the CCF, but it is a standard that is open to the entire Java community.

1.2.3 External Security Interface

The ESI is used for verifying and changing the user ID and password information held in the CICS external security manager (ESM), such as RACF. It is based on the CICS Password Expiration Management (PEM) function. For further details on programming with the ESI, refer to Chapter 4 “ECI and ESI applications”, in the redbook *Java Connectors for CICS*, SG24-6401.

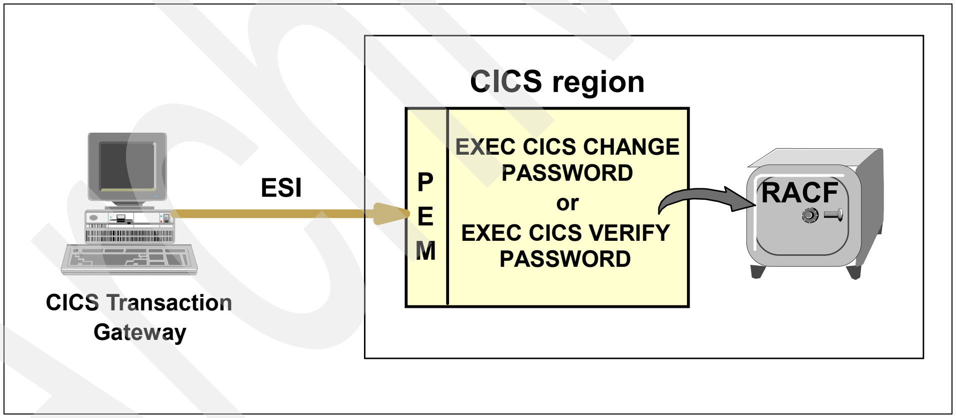


Figure 1-8 External Security Interface

ESI calls to CICS can only be made using the `ESIRequest` class provided by the CICS TG base classes. This class provides a Java interface to the ESI, and provides two simple PEM requester functions:

Verify Password Allows a client application to verify that a password matches the password for a given user ID stored by the CICS ESM.

Change Password Allows a client application to change the password held by the CICS ESM for a given user ID.

There is no other interface available for the ESI, although both the EPI and ESI allow user IDs and passwords to be flowed within the actual requests. In this case the user ID and password will be either authenticated within CICS if using a distributed CICS Transaction Gateway, or within the CICS TG if using the CICS TG for z/OS.

Part 2

Configuring CICS connections

In this section, we discuss how we configured the CICS TG Client daemon on Windows 2000 to connect to a CICS region using either the TCP/IP, APPC, or TCP62 protocols.

For additional information on configuring the Client daemon or CICS Universal Client, refer to the CICS Universal Client configuration pamphlets available at:

`http://www-3.ibm.com/software/ts/cics/library/manuals/index40.html`

2

APPC connections to CICS

In this chapter, we describe how we configured an APPC connection on an SNA network from the CICS Transaction Gateway (CICS TG) V5.0 on Windows 2000 to our CICS Transaction Server (CICS TS) V2.2 region. The following platforms and products are supported for APPC communication over an SNA network:

- Windows operating systems
  - Any one of:
    - IBM eNetwork Communications Server V6.0.1
    - IBM eNetwork Personal Communications V5.01
    - Microsoft SNA Server V4.0 + SP3
- AIX
  - IBM eNetwork Communications Server V5.0.2 & V6.01

Our configuration is illustrated in Figure 2-1 on page 18.

2.1 Introduction to APPC

If two systems are to communicate successfully, they must use a common set of rules that both understand. A communications protocol is a set of rules that defines, for example, a set of standard requests and responses, and the order in which they can be sent.

LU TYPE 6.2 (LU 6.2) is a Systems Network Architecture (SNA) protocol that supports both system-to-system communication and system-to-device communication. LU 6.2 is also known as advanced program-to-program communications (APPC).

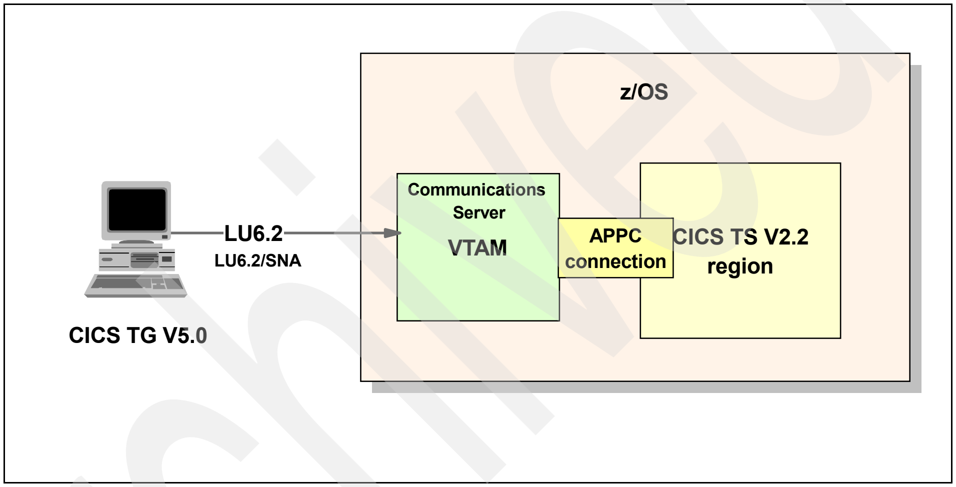


Figure 2-1 CICS client-server connections

For all connected client systems, CICS requires both a CONNECTION and SESSIONS definition. The CONNECTION definition identifies the remote system, and one or more SESSIONS definitions are associated with this CONNECTION to define the properties of the sessions.

2.1.1 Software checklist

We used the levels of software shown in Table 2-1.

Table 2-1 Software levels, TCP62

Client workstation	z/OS
Windows 2000 Service Pack 2	z/OS V1.2
CICS Transaction Gateway for WIndows V5.0	CICS Transaction Server V2.2
IBM Personal Communications V5.0	
IBM Java Development Kit V1.3.0	

2.1.2 Definitions checklist

The definitions we used in this scenario are summarized in Table 2-2. Before you configure the products, we recommend that you acquire definitions for each parameter listed. Reference keys shown in the Key column are assigned to definitions that must contain the same value in more than one product. The Example column shows the value we used.

Table 2-2 Definitions checklist for APPC

Key	VTAM®	CICS TS region	Personal Communications	CICS TG	Example
1	NETID		First part of partner LU name		USIBMSC
2	PU		CP Alias		SC02009
3	LU	NETNAME	Local LU name	Local LU name	SC02009I
4	XID (IDBLK + IDNUM)		Local Node ID		05D02009
5	VTAM comms. controller MAC address		Destination address		400022160011
6	APPL	APPLID	Second part of partner LU name	Partner LU name	SCSCPJA1
7	LOGMODE	MODENAME		Mode name	APPCMODE
8	SSCP		Fully qualified CP name		USIBMSC.SC38M

The following are the values required in our configuration:

- **1 NETID - USIBMSC**

  The NETID is the VTAM SNA network name. It is defined for your VTAM network in the VTAM start options.

- **2 PU — SC02009**

  The PU SC02009 is the name of our workstations physical unit (PU). It is named in the VTAM switched major node as shown in 2.3.3, “VTAM switched major node” on page 27. We also used the same name for the workstation control point (CP).

- **3 LU — SC02009I**

  The LU SC02009I is the independent LU 6.2 used by the CICS Transaction Gateway. It must also be defined to VTAM in a switched major node.

- **4 XID — 05D02009**

  The XID (or node ID) is configured in the VTAM switched major node using IDBLK and IDNUM. It is used in the XID exchange to activate the link station.

- **5 MAC Address — 400022160011**

  This is the MAC address of the communications adapter used by the VTAM front-end processor. This might by a communications controller such as the 3172, or a 2216 router, or an OSA adapter.

- **6 APPL — SCSCPJA1**

  This is the CICS APPLID and also the VTAM APPL. It is defined in the VTAM application major node (see 2.3.1, “VTAM application major node” on page 26) used by the CICS region.

- **7 LOGMODE — APPCMODE**

  This the mode group used to control LU 6.2 session properties. It must be defined in a VTAM logon mode table, which must be named on the VTAM APPL definition as shown in 2.3.2, “VTAM logon mode table” on page 27.

- **8 SSCP — SC38M**

  This the CP for the PU of the VTAM front-end processor. In VTAM, it is referred to as the SSCP.

2.2 APPC connections in CICS

To define APPC connections from CICS, you will need to:

- Specify the SIT parameter: ISC=YES
- Install CSD groups: DFHCLNT and DFHISC
- Create and install CONNECTION and SESSIONS definition, or configure CICS connection autoinstall

2.2.1 CICS CONNECTION definitions

The connection definition defines the location of the remote system and the parameters of the connection to it. The connection we used for the logical unit (LU) is shown in Figure 2-2.

```
OVERTYPE TO MODIFY                                         CICS RELEASE = 0620
CEDA  DEFine CONnection( SC09 )
 CONnection     : SC09
 Group          : SC02009
 DEscription  ==>
CONNECTION IDENTIFIERS
 Netname      ==> SC02009I
 CONNECTION PROPERTIES
 ACcessmethod ==> Vtam              Vtam | IRc | INdirect | Xm
 PRotocol     ==> Appc              Appc | Lu61 | Exci
 Conntype     ==>                   Generic | Specific
 SIngless     ==> No                No | Yes
OPERATIONAL PROPERTIES
 AUtoconnect  ==> Yes               No | Yes | All
 INService    ==> Yes               Yes | No
SECURITY
 SEcurityname ==>
 ATtachsec    ==> Local             Local | Identify | Verify | Persistent

                                                  SYSID=PJA1 APPLID=SCSCPJA1
```

Figure 2-2 CICS APPC CONNECTION definition

NETNAME Specifies the LU name of the partner LU 6.2. In our example this is (3) SC02009I, the LU used by our CICS TG.

ACCESSMETHOD For advanced program-to-program communication (APPC) connections to the CICS Transaction Gateway, ACCESSMETHOD must be set to VTAM.

PROTOCOL Must be specified as APPC.

SINGLESESS Should be set to NO if using the CICS TG, since they require the use of LU 6.2 parallel sessions.

AUTOCONNECT Specifies if CICS is to bind sessions (drive CNOS) when the connection is installed. We set this parameter to YES.

ATTACHSEC Defines the settings for SNA LU 6.2 conversation level security. If you wish to flow a user ID and password from the CICS Transaction Gateway, this parameter should be set to VERIFY. If you do not wish to flow a user ID, this should be set to LOCAL. No other setting is valid for the CICS TG.

2.2.2 CICS SESSIONS definitions

For each CICS CONNECTION definition, you need to define one or more SESSIONS definitions to define the SNA mode groups to be used within that connection. Figure 2-3 illustrates the sessions definition we used to define the mode group APPCMODE for our APPC connection SC09 shown in Figure 2-3.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0620
CEDA  DEFine Sessions( SC09     )
 Sessions       : SC09
 Group          : SC02009
 DEscription  ==>
SESSION IDENTIFIERS
 Connection   ==> SC09
 SESSName     ==>
 NETnameq     ==>
 MOdename     ==> APPCMODE
SESSION PROPERTIES
 Protocol     ==> Appc               Appc | Lu61 | Exci
 MAximum      ==> 008 , 001          0-999
 RECEIVEPfx   ==>
 RECEIVECount ==>                    1-999
 SENDPfx      ==>
 SENDCount    ==>                    1-999
 SENDSize     ==> 04096              1-30720
 RECEIVESize  ==> 04096              1-30720
 SESSPriority ==> 000               0-255
OPERATIONAL PROPERTIES
 Autoconnect  ==> Yes               No | Yes | All

                                                   SYSID=PJA1 APPLID=SCSCPJA1
```

Figure 2-3 CICS APPC SESSIONS definition

CONNECTION Specifies the name of the associated CONNECTION definition.

MODENAME Specifies the mode group as defined in a VTAM LOGMODE. The MODENAME must be unique among the SESSIONS definitions related to one CONNECTION definition. In our example it is (7) APPCMODE.

PROTOCOL Must be specified as APPC.

MAXIMUM Specifies the maximum number of sessions that are to be supported. The first value is the maximum number of sessions that can be supported. The second value specifies the number of contention-winner sessions (CICS-owned sessions). These values are negotiated when the sessions are actually bound (during change number of sessions [CNOS] flows), the negotiated values will depend on the settings specified in the partner SNA node.

When using the CICS TG, the number of sessions should be at least as big as the `MAXREQUESTS` parameter in the CTG.INI file to prevent this becoming a potential bottleneck to throughput. The number of contention-winner sessions should be set to 1 to ensure that START requests are shipped serially from the server to the client.

AUTOCONNECT Determines if the sessions for this mode group will be bound when the connection is installed. YES specifies that the contention-winner sessions are to be bound, and ALL specifies that the both contention-winner and contention-loser sessions are to be bound.

2.2.3 CICS connection autoinstall

CICS connection autoinstall, like terminal autoinstall, allows CONNECTION and SESSIONS definitions to be dynamically created (and deleted) based on a sample template definition.

Configuring connection autoinstall is a relatively simple exercise. First, update the default autoinstall program from DFHZATDX to DFHZATDY. This is achieved by specifying the SIT parameter AIEXIT=DFHZATDY. The new autoinstall program DFHZATDY has the same function as DFHZATDX, but supports autoinstall of connections as well as terminals. Alternatively, it is possible to write your own autoinstall user-replaceable module based on the samples provided. Autoinstall of connections is particularly useful when dealing with many similar connections, or when you are unsure of the LU names (netnames) to be used. This will be the case if you are using dynamic LUs with TCP62 connections from the CICS TG.

Like autoinstall for terminals, autoinstall for APPC connections requires model definitions. The supplied DFHZATDY autoinstall program uses the template CBPS. This is supplied in DFHAI62 group and must be copied to your own group and modified accordingly. Figure 2-4 and Figure 2-5 on page 25 illustrate the CBPS connection and session templates we used during this project.

```
OVERTYPE TO MODIFY                                         CICS RELEASE = 0620
CEDA  ALter CONnection( CBPS )
 CONnection     : CBPS
 Group          : PJAAI62
 DEscription  ==> APPC Autoinstall template for parallel session secondary
CONNECTION IDENTIFIERS
 Netname      ==> TMPLATE1
 INDsys       ==>
CONNECTION PROPERTIES
 ACcessmethod ==> Vtam                Vtam | IRc | INdirect | Xm
 PRotocol     ==> Appc                Appc | Lu61 | Exci
 SIngleSess   ==> No                  No | Yes
OPERATIONAL PROPERTIES
 AUtoconnect  ==> Yes                 No | Yes | All
 INService    ==> Yes                 Yes | No
SECURITY
 SEcurityname ==>
 ATtachsec    ==> Local               Local | Identify | Verify | Persistent
                                     | Mixidpe
                                               SYSID=PJA1 APPLID=SCSCPJA1
```

Figure 2-4 CICS autoinstall connection template

The parameters in the CONNECTION definition template are essentially the same as when defining a static definition (2.2.1, "CICS CONNECTION definitions" on page 21), except that the NETNAME is not required.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0620
 CEDA  ALter Sessions( CBPS     )
  Sessions       : CBPS
  Group          : PJAAI62
  DEscription  ==> APPC Autoinstall template for parallel session secondary
 SESSION IDENTIFIERS
  Connection   ==> CBPS
  SESSName     ==>
  NETname      ==>
  MOdename     ==> APPCMODE
 SESSION PROPERTIES
  Protocol     ==> Appc              Appc | Lu61 | Exci
  MAximum      ==> 008 , 001         0-999
  SENDSize     ==> 04096             1-30720
  RECEIVESize  ==> 04096             1-30720

                                                    SYSID=PJA1 APPLID=SCSCPJA1
```

Figure 2-5 CICS autoinstall sessions template

The parameters in the SESSIONS definition template should also be the same as when defining a static definition (see 2.2.2, "CICS SESSIONS definitions" on page 22), except that the CONNECTION parameter should refer to the CBPS CONNECTION definition.

If you use the supplied connection autoinstall program (DFHZATDY), the connection name generated will be based on the last four characters of the NETNAME. If you need to change this, then you will have to create your own user-replaceable module from the sample provided in CICSTS22.CICS.SDFHSAMP.

2.3 APPC connections and VTAM

CICS uses the services of VTAM for all SNA network connectivity to other LUs, including the CICS Transaction Gateway. CICS itself acts as an LU 6.2 and requires a connection with parallel sessions to any connected CICS Transaction Gateway. To use LU 6.2 parallel sessions with CICS, you need to configure the following VTAM resources:

- **Application major node:** This contains the CICS LU definition (or APPL).
- **Logon mode table:** This contains the LOGMODE, which defines the characteristics of the LU 6.2 sessions to be used.
- **Switched major node:** This contains details of the connected node, including the physical unit (PU) and associated partner LUs.

If you are using an LU 6.2 over an IP connection (as provided by the TCP62 protocol), you do not require a switched major node but instead will require a TCP major node, and possibly CDRSC definitions. For further details, refer to 3.3, “VTAM definitions” on page 44.

- **Network name:** The first parameter you should ascertain before you define any VTAM resource is the name of your VTAM network, which will most likely have already been defined by your VTAM systems programmer. This is defined in the NETID parameter in the VTAM start procedure; in our configuration, it was (1) USIBMSC.

In the following sections, we detail how to configure each of these resources.

2.3.1 VTAM application major node

For a CICS system to be able to communicate with other SNA logical units using VTAM services, you must define CICS to ACF/VTAM. This is achieved by defining a VTAM application program (APPL) major node for each CICS region. These are defined as APPL definition statements in a member of the SYS1.VTAMLST library. Example 2-1 shows the VTAM APPL definition for our CICS region (6)SCSCPJA1.

Example 2-1 VTAM APPL definition

```
         VBUILD TYPE=APPL
SCSCPJA1 APPL AUTH=(ACQ,VPACE,PASS),VPACING=0,EAS=5000,PARSESS=YES,    X
              SONSCIP=YES,APPC=NO,MODETAB=MTAPPC
```

AUTH Must be specified as ACQ to allow CICS to acquire sessions. VPACE is required to allow pacing of the LU 6.2 flows. PASS will allow existing terminal sessions to be passed to other CICS region using the CICS command EXEC CICS ISSUE PASS.

EAS Is the number of network addressable units (NAU) that this LU can establish sessions with, and can have a maximum value of 5000. An NAU is either a logical unit, a physical unit, or a control point. This parameter therefore limits the maximum number of LU-LU sessions any one CICS region can have.

PARSESS Must be specified as YES, since the CICS TG requires LU 6.2 parallel sessions to be used.

SONSCIP Specifies session outage notification support, which enables CICS, in some cases, to recover a failed system without requiring operator intervention.

APPC Must be specified as NO.

MODETAB Is the name of the VTAM logon mode table that contains the customized user mode entries. You may omit this operand if you choose to add your MODEENT entries to the IBM-supplied logon mode table, ISTINCLM, or if you use only the predefined entries (such as #INTER) from this table.

2.3.2 VTAM logon mode table

CICS requires a VTAM logon mode table for every mode used in a CICS SESSIONS definition. VTAM logon modes are defined in a LOGMODE. Figure 2-2 shows the logon mode table we used to define the mode (7) APPCMODE used for the LU 6.2 sessions to our CICS region SCSCPJA1.

Example 2-2 VTAM LOGMODE, APPMODE

```
        *      LOGMODE ENTRY FOR CICS LU 6.2 PARALLEL SESSIONS**
POKMODE   MODETAB
APPCMODE MODEENT LOGMODE=APPCMODE,
    MODEEND
    END
```

The session limits are not defined in this logmode table, since they are defined in the CICS CONNECTION definition (Figure 2-3 on page 22). You should also note that the SNASVCMG mode is supplied by VTAM for the management of LU 6.2 parallel sessions, and should not be used for normal APPC traffic, including CICS flows.

2.3.3 VTAM switched major node

The VTAM switched major node defines both the link station to the remote PU and the LUs available within that PU. In VTAM all partner LU 6.2s must either be defined in VTAM, or defined dynamically (specify DYNLU=YES). However, if your VTAM network is a subarea network (that is, not in an Advanced Peer-to-Peer Networking [APPN] network) and does not use dynamic LUs, then you will need to define the partner LU 6.2s using a switched major node. Example 2-3 illustrates the switched major node we used to define the SNA node (2) SC02009 and the independent LU 6.2 (3) SC02009I. We also define here the XID (4) that is made of the IDBLK (05D) and IDNUM (02009).

Example 2-3 VTAM PU, SC02009I

```
       SC02009  PU    ADDR=01,                                          +
               IDBLK=05D,IDNUM=02009,                                   +
               ANS=CONT,DISCNT=NO,                                      +
               IRETRY=NO,ISTATUS=ACTIVE,                                +
               MAXDATA=265,MAXOUT=7,                                    +
```

```
               MAXPATH=1,USSTAB=USSRDYN,                           +
               PUTYPE=2,SECNET=NO,                                 +
               MODETAB=POKMODE,DLOGMOD=DYNTRN,                     +
               PACING=1,VPACING=2
*
SC02009I LU    LOCADDR=0,DLOGMOD=APPCMODE
```

2.4 Configuring IBM Personal Communications

To configure IBM Personal Communications, we selected **Start -> Programs -> IBM Personal Communications -> SNA Node Configuration**. The Node Configuration window is the main window and from here we chose the following:

- Configure the Node and enter the following information:
  - Fully qualified CP name(1.2): USIBMSC.SC02009
  - CP alias (2)): SC02009
  - Local Node ID(4):
    - Block ID: 05D
    - Physical Unit ID: 02009
- Configure a LAN Device and enter the following information:
  - Accept the defaults
- Configure the a LAN Connection and enter the following information:
  - Destination Address(5): 400022160011
- Configure a Local LU 6.2 definition:
  - Local LU name and alias(3): SC02009I
- Configure a Partner LU 6.2 definition as follows:
  - Partner LU name(1.6): USIBMSC.SCSCPJA1
  - Partner LU alias(6)): SCSCPJA1
  - Fully qualified CP name(8): USIBMSC.SC38M
- Configure a Mode as follows:
  - Mode name(7): APPCMODE
  - PLU mode session limit: 8
  - Minimum contention winner sessions: 99
  - Advanced settings:
    - Maximum negotiable session limit: 100
    - Receive pacing window size: 8
    - Class of service name: #CONNECT
    - Use default RU size

- Configure a Transaction Program definition as follows:
  - TP name: CRSR
  - Complete path name: C:\Program Files\IBM\IBM CICS Transaction Gateway\bin\cclclnt.exe
  - Program parameters: CRSR
  - Conversation type: Mapped
  - Synchronization level: Any
  - Conversation Security: Not checked
  - Advanced settings:
    - Receive_Allocate timeout: 0
    - Incoming allocate timeout: 0
    - TP instance limit: 1
    - Make sure none of the boxes are checked

The file was saved under the \private subdirectory with the name sc02009.acg.

Configuration booklets, which document the required configuration for all the different CICS TG scenarios, can be found at the following Web site:

`http://www-3.ibm.com/software/ts/cics/library/manuals/ctg40dl.html`

2.5 APPC definitions for the CICS TG

We used the CICS TG Configuration Tool to configure the connection to the CICS region. The Configuration Tool generates the CTG.INI file. The Client daemon uses the CTG.INI file to establish a connection to a CICS region.

We defined the server connection to our CICS TS region as shown in Figure 2-6 on page 30.

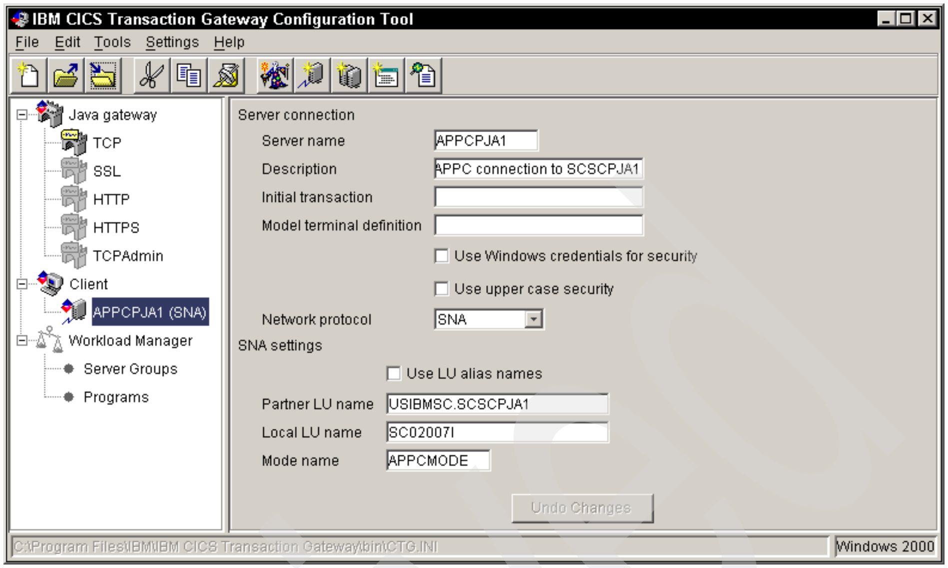


Figure 2-6 Configuration Tool, APPC definitions

The parameters that we used are as follows:

- **Server**

  An arbitrary name for a particular CICS region (we used the value APPCPJA1) to distinguish this connection from our TCP62 and TCP/IP connections.

- **Description**

  An arbitrary description for the CICS region.

- **Network Protocol**

  The protocol for communication with the CICS region (in this case, SNA).

- **Partner LU name**

  The LU name of the server as it is known to the SNA configuration (in our case, the fully qualified partner LU name of (1.6) USIBMSC.SCSCPJA1).

- **Local LU name**

  The name of a local LU to be used when connecting to the server (in our case, the local LU name of (3) SC02009I).

- **Mode name**

  The LU 6.2 mode name to be used when connecting to the server (in our case (7) APPCMODE).

Testing the configuration

After we installed and configured the software components, we tested our configuration as follows:

1. Checked the SNA Link Station from the workstation to VTAM was active, as follows:

   – Start the SNA Node Operation tool by clicking **Start -> Programs -> IBM Personal Communications -> Administrative and PD Aids -> SNA Node Operations.**

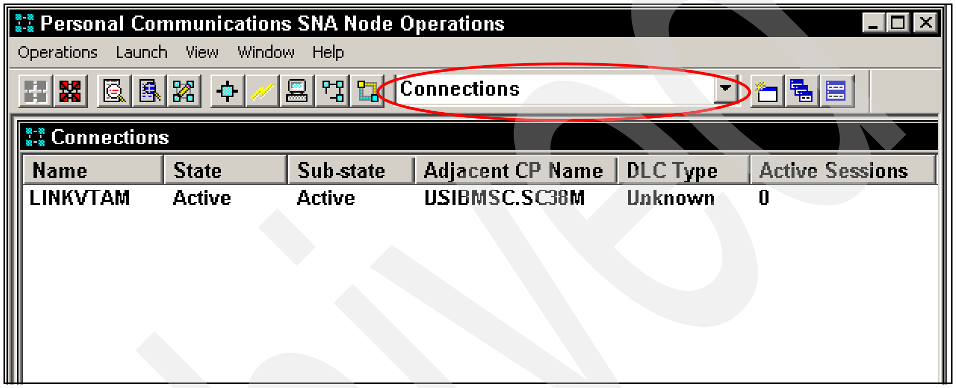


Figure 2-7 SNA Node Operations, Connection status

   – From the central drop-down list box, select **Connections**. Check that the state of the connection is Active. This indicates that the XID exchange between the workstation PU and the VTAM PU was successful.

2. Started the CICS region SCSCPJA1.
3. Installed the CICS CONNECTION and SESSIONS definitions (since we were not using connection autoinstall).
4. Checked the state of the CICS connection using the command:

```
CEMT I CONN
```

5. The connection should show Acq, meaning LU 6.2 sessions are bound.

```
 I CONN
STATUS:  RESULTS - OVERTYPE TO MODIFY
  Con(SC09) Net(SC02009I)     Ins Acq Vta Appc
                  Nqn(USIBMSC.SC02009I )










                                                   SYSID=PJA1 APPLID=SCSCPJA1
 RESPONSE: NORMAL                             TIME:  17.37.13  DATE: 07.22.02
PF 1 HELP       3 END       5 VAR        7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Figure 2-8 CICS APPC connection definition

6. Checked the state of the LU 6.2 sessions in the Personal Communications SNA Node Operations tool, as follows:
   - Start the SNA Node Operation tool using **Start -> Programs -> IBM Personal Communications -> Administrative and PD Aids -> SNA Node Operations.**
   - From the central drop-down list box, select **LU 6.2 Sessions** (see Figure 2-9). Each active session will be individually listed. In our configuration we had one SNASVCMG session bound, and one APPCMODE session bound.

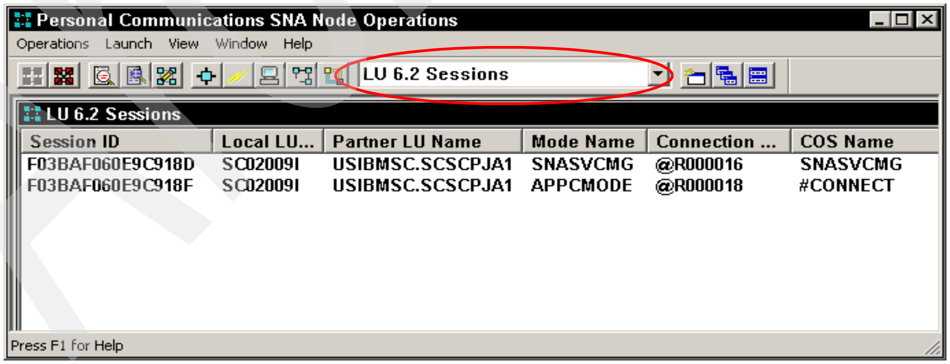


Figure 2-9 SNA Node Operations, Connection status

7. Started the Client daemon connection to the CICS region using the command:

   `CICSCLI /S=APPCPJA1`

   APPCPJA1 is the server name specified in the Client configuration as shown in Figure 2-6 on page 30.

8. Checked the status of the connection to the CICS region using the command:

   `CICSCLI /L`

   The connection status is available, as shown in Example 2-4.

Example 2-4 Checking connection status

```
C:\>CICSCLI /L
CCL8001I CICSCLI - CICS Client Control Program
CCL0002I (C) Copyright IBM Corporation 1994,2001. All rights reserved.
CCL8041I The CICS Client is using the following servers:
CCL8042I Server 'APPCPJA1' (using 'SNA' to 'USIBMSC.SCSCPJA1') is available
```

9. Issued the `CICSTERM /S=APPCPJA1` command to install a client terminal on the CICS region.

10. Ran the CICS-supplied transaction CEOT to display the name of our installed terminal in CICS (Figure 2-10).

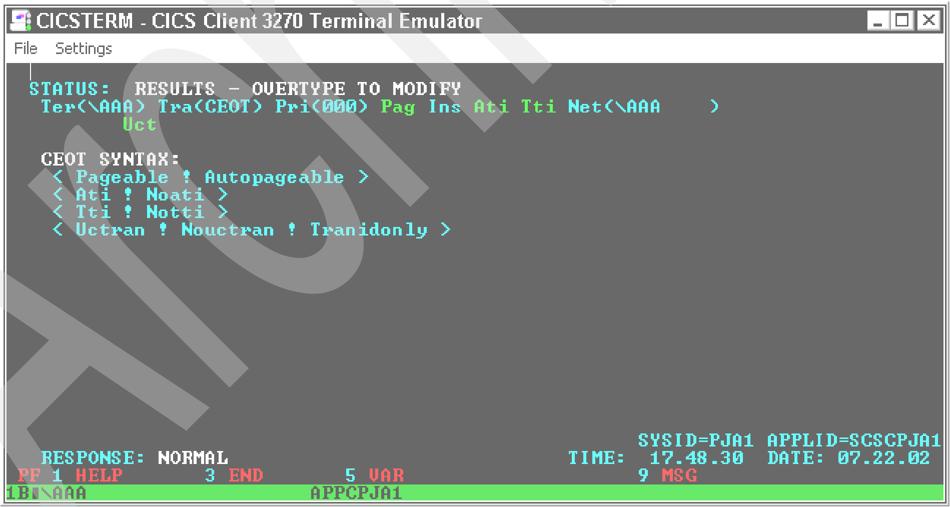


Figure 2-10 CICSTERM, CEOT

Lastly, we checked the CICS region CSMT log to view the messages from the CICS connection autoinstall as shown in Example 2-5.

Example 2-5 CICS log messages from the connection autoinstall

```
DFHZC3461 I 07/22/2002 17:40:21 SCSCPJA1 -AAY CSNE Node SC02009I session started.  ((1) Module
name: DFHZOPX)  NQNAME   -AAY,CSNE,17:40:21,USIBMSC  SC02009I
DFHZC4900 I 07/22/2002 18:04:34 SCSCPJA1 -AAY CLS1 CNOS received from Node SCHPHIL9 System SC09
Modename APPCMODE, Max =  8, Win=7, successful. ((1) Module name: DFHZGCN)
DFHZC4901 I 07/22/2002 17:40:21 SCSCPJA1 -AAY CLS1 Node SC02009I System SC09 Modename APPCMODE,
Negotiated values:  Max=8, Win=7. ((1) Module name: DFHZGCN)
DFHZC3461 I 07/22/2002 17:40:22 SCSCPJA1 -AA6 CSNE Node SC02009I session started.  ((1) Module
name: DFHZOPX) NQNAME   -AA6,CSNE,17:40:22,USIBMSC  SC02009I
DFHZC5966 I 07/22/2002 17:48:27 SCSCPJA1 INSTALL started for TERMINAL (  \AAA) (Module name:
DFHBSTZ).
```

2.6 Problem determination

In this section, we discuss the different tools available for performing problem resolution for LU 6.2 connections.

2.6.1 Tips and utilities

The CICS transaction CEMT can be used to query the status of CICS connections and sessions.

To view the status of all APPC connections, issue the command:

```
CEMT INQ CONN
```

To view the status of LU 6.2 sessions, issue the command:

```
CEMT INQ MOD CONN(SC09)
```

This will provide the netname. Then use the following command to see the individual status of each session:

```
CEMT INQ NETNAME(SC02009I)
```

SNA sense codes

We found the Personal Communications GetSense utility helpful for quickly looking up SNA sense code. To launch GetSense from Windows, click **Start -> Programs -> IBM Personal Communications -> Administrative and PD Aids** and click **Display SNA Sense Data**. The GetSense window will appear, as shown in Figure 2-11.

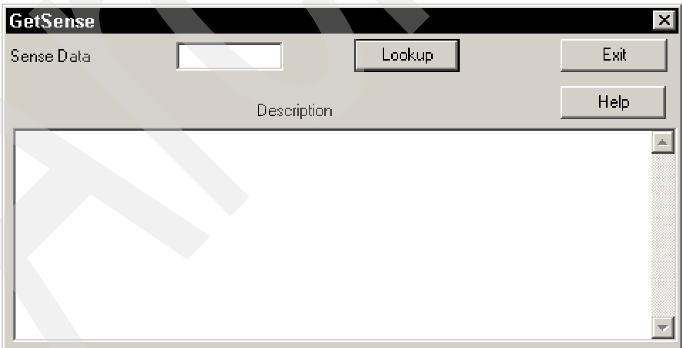


Figure 2-11 GetSense window

SNA sense data can be entered in the Sense Data field and a detailed sense code description displayed by clicking **Lookup**.

2.6.2 Tracing

In this section, we provide information on how to use the trace facilities provided with the Client daemon to debug a simple APPC communications problem.

Client daemon

The first place to look for error messages when using the CICS TG on the Windows system is the Client daemon log file. This is specified in the Client configuration parameters in the CICS TG Configuration Tool. We used the default name CICSCLI.LOG, which can be found in the CICS TG \bin subdirectory.

The first step to useful tracing with the Client daemon is identifying which components should be traced. A full list is shown in Example 2-6 and can be viewed using the command `CICSCLI /M`. If in doubt, you should trace with the default selected components, as these have been selected to be sufficient to diagnose most problems. The default components have an `X` in front of them in the following example. To turn on all the parameters, you can use `/M=ALL` option.

Example 2-6 CICS TG client trace options

```
C:\>cicscli /m
CCL8001I CICSCLI - CICS Client Control Program
CCL0002I (C) Copyright IBM Corporation 1994,2001.  All rights reserved.
CCL1120I Trace has not been started
CCL1100I The following list shows what components can be traced
CCL1101I An 'X' indicates which components are enabled
CCL1102I        CICSCLI Command Process      [CLI]
CCL1103I        Interprocess Communication   [TRN]
CCL1104I    X   Protocol Drivers (Level 1)   [DRV.1]
CCL1105I        Protocol Drivers (Level 2)   [DRV.2]
CCL1106I    X   API (Level 1)                [API.1]
CCL1107I        API (Level 2)                [API.2]
CCL1108I    X   Client Daemon                [CCL]
CCL1109I        Terminal Emulators           [EMU]
CCL1111I        C++ Class Libraries          [CPP]
CCL1112I        Workload Manager             [LMG]
CCL1113I        CICS Client Service for NT   [SER]
```

Trace is written to a binary file called, by default, CICSCLI.BIN. To read the trace, you must format the binary file into a text file by using the `CICSFTRC /D` command. This will produce a viewable file called CICSCLI.TRC. Both CICSCLI.BIN and CICSCLI.TRC are found, by default, in the \BIN subdirectory.

The CICS TG Client daemon trace is started with the `CICSCLI /D` command. If you need to trace the client from startup, you can specify all the parameters needed together on the command line, such as:

```
CICSCLI /S=SCSCPJA1 /D=9999 /M=ALL
```

For more information on Client daemon tracing, wrapping trace, and formatting options, refer to *CICS Transaction Gateway Windows Client Administration,* SC34-5940.

IBM Personal Communications

If you suspect there is an SNA communication problem in your configuration, then using IBM Personal Communications Trace Facility can provide useful diagnostic information. The most useful type of SNA trace is likely to be the data-link control (DLC) level tracing (or in SNA terms, link station tracing), which will show you the binds, bind responses, attaches (FMH-5), and attach errors (FMH-7).

To start the trace utility, click **Start -> Programs -> IBM Personal Communications -> Administrative and PD Aids -> Trace Facility**. This will display the Trace Facility as shown in Figure 2-12.

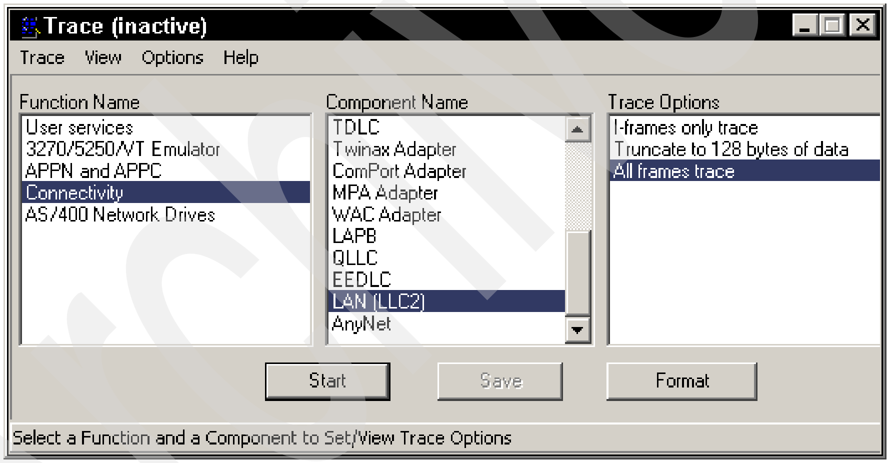


Figure 2-12 Personal Communications Trace Facility

If you have a LAN connection and wish to activate DLC tracing, select **Connectivity** in the left pane, then **LAN (LLC2)** in the middle pane, then **All frames trace** in the right pane. To activate the trace, click **Start**, run the desired operation, and then click **Stop**. To save the trace to a file, click **Save** and you will be prompted for a file name and whether or not to append or replace the trace file. By default, an unformatted trace file will be saved as NSTRC.TRC. Then to format the trace file, click **Format**, and the formatted file NSTRC.TLG will be produced. This formatted trace file can then be viewed using the Personal Communications Log viewer utility by double-clicking the file.

Interpretation of these traces is beyond the scope of this book. For further details you should reference *Systems Network Architecture Formats*, GA27-3136-19. However, although SNA traces can be complicated to fully understand, they can also be useful to an inexperienced user, by revealing the actual LU name, partner LU name, mode name, and other parameters that actually were used. In addition, a simple inspection can often reveal the data contained with the request unit (RU), which will equate to the CICS COMMAREA or 3270 data stream sent to or returned from a CICS program or transaction.

3

TCP62 connections to CICS

In this chapter, we describe how we configured a TCP62 connection from the CICS Transaction Gateway (CICS TG) V5 on Windows 2000 to our CICS Transaction Server (CICS TS) V2.2 region. We show only Windows 2000, but these steps can be used to configure all the platforms except z/OS, since there are no major differences. Our configuration is illustrated in Figure 3-2 on page 41.

3.1 Introduction to TCP62

In the CICS TG V5 the TCP62 protocol is now supported on all platforms, except for z/OS. It is integrated with the base product and it is not a separately installable feature. Configurations and definitions used with the previous (CICS TG V3) implementation will work with the new implementation transparently.

TCP62 is a communications mechanism that provides the ability to connect from the CICS TG to a CICS TS region over an IP network. It utilizes the function of IBM's Multiprotocol Transport Networking (MPTN) protocol switching technology to send LU 6.2 (APPC) Systems Network Architecture (SNA) flows over an IP network. Figure 3-1 on page 45 illustrates how APPC, SNA, and LU 6.2/IP work together at the protocol level.

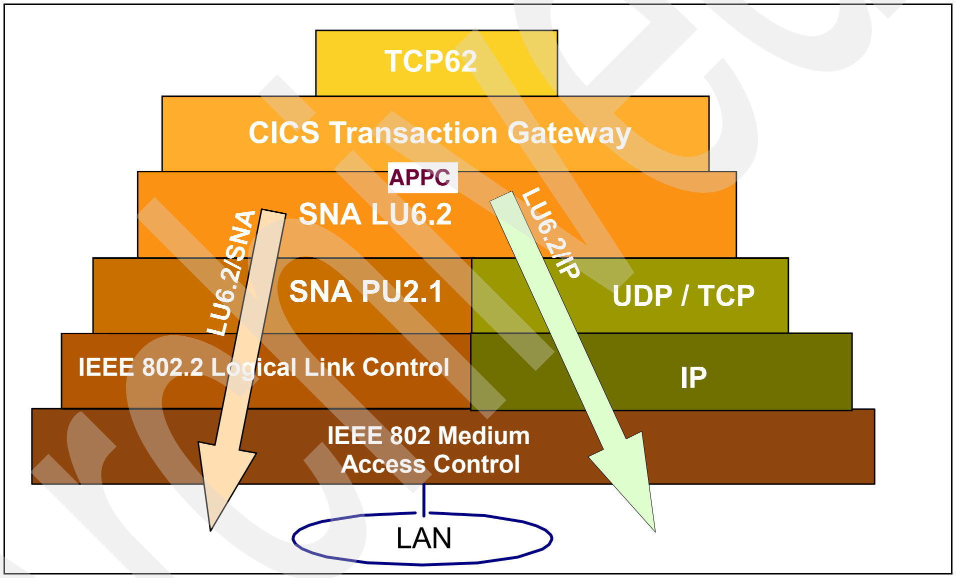


Figure 3-1 TCP62 and SNA protocols

The TCP62 SNA node on the CICS TG machine is dynamically configured using parameters from the configuration file (CTG.INI). Therefore no SNA configuration is necessary on the Windows client. On z/OS it is necessary to configure both an APPC connection in CICS and the required AnyNet® definitions for VTAM.

Static LU names

We simplified our configuration by using static definitions. With this method the LU names are pre-set and not generated dynamically. This is useful if you only have a few Client workstations or in situations where you might want to match the LU name (NETNAME) with a specific connection in CICS, for security, audit, or other reasons.

Dynamic LU name generation

It is also possible to use dynamic LU name generation with TCP62 connections. This reduces the amount of configuration required and has the added benefit of allowing the same CTG.INI configuration file to be used on all the workstations in your TCP/IP subnet. For more details refer to "TCP62 dynamic LU names" on page 48.

In Figure 3-2, we illustrate an overview of our TCP62 environment that shows how we configured in this chapter.

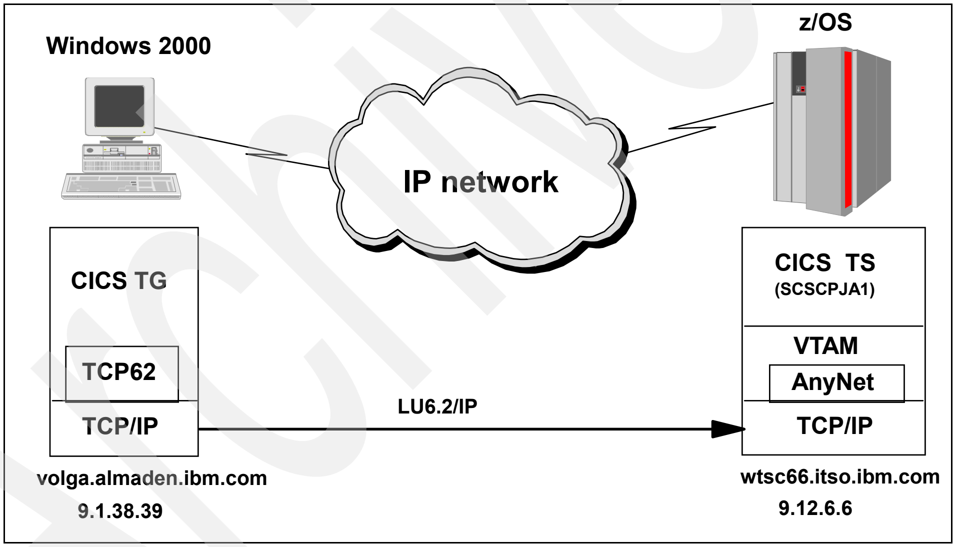


Figure 3-2 CICS TG TCP62 configuration

> **Tip:** We also found it was possible to move the CTG.INI file to the AIX platform without making any modifications, and use our TCP62 configuration to connect our CICS TG for AIX to CICS. However, on some platforms the protocol drivers have different names (for more information see 9.3, "Testing the configuration" on page 224). To avoid this, the **`ctgcfg`** command can be run with the **`-plat`** option, specifying the particular platform required. The list of possible platforms can be queried using the command **`ctgcfg -?`**.

3.1.1 Software checklist

We used the levels of software shown in Table 3-1.

Table 3-1 Software levels, TCP62

Client workstation	z/OS
Windows 2000 Service Pack 2	z/OS V1.2
CICS Transaction Gateway for WIndows V5.0	CICS Transaction Server V2.2
IBM Java Development Kit V1.3.0	

3.1.2 Definitions checklist

The definitions we used to configure this scenario are summarized in Table 3-2. Before you configure the products, we recommend that you decide on definitions for each parameter listed. Reference keys shown in the Key column are assigned to definitions that should contain the same value in more than one product. The Example column shows the values we used in our configuration.

Table 3-2 Definitions checklist for TCP62

Key	z/OS Comms. Server (VTAM and TCP/IP)	CICS TS region	CICS TG (TCP62)	Example
1	IP address			9.12.6.6
2	DNSUFX		AnyNet domain name suffix	itso.ibm.com
3	PORT		397	397
4	NETID		Partner LU name	USIBMSC
5	APPL	APPLID	Partner LU name	SCSCPJA1
6	LOGMODE	MODENAME	Mode name	APPCMODE
			IP address mask for LU name template	00000000
			Fully qualified control point name	USIBMSC.CCLI62CP
		NETNAME	Local LU name	CCLI62LU

3.2 APPC connections in CICS

From a CICS perspective, the TCP62 client is just another LU 6.2 independent LU. Thus, you will need to configure ISC support in CICS, and if you use dynamic LUs, you will need to configure autoinstall of CICS connections, since it will not be possible to predict the LU name of the client LU.

The VTAM APPLID 5 of our CICS region was defined as SCSCPJA1. This is combined with the VTAM NETID 4 of USIBMSC to make a fully qualified partner LU name of USIBMSC.SCSCPJA1.

We also performed the following configuration in our CICS region, depending on whether we were using autoinstalled connections of static connections.

Static connections

We did the following with our static connections:

- Configured the following SIT parameters:
  – ISC=YES
  – APPLID=SCSCPJA1
- Installed groups: DFHISC and DFHCLNT
- Defined and installed CICS CONNECTION and SESSIONS definitions.

For more details refer to 2.2, “APPC connections in CICS” on page 21.

Autoinstalled connections

We did the following with our autoinstalled connections:

- Configured the following SIT parameters:
  – ISC=YES
  – APPLID=SCSCPJA1
  – AIEXIT=DFHZATDY
- Installed groups: DFHISC and DFHCLNT
- Enabled CICS APPC CONNECTION autoinstall for parallel sessions

For more details on configuring autoinstalled APPC connections in CICS, refer to 2.2.3, “CICS connection autoinstall” on page 23.

3.3 VTAM definitions

In this section, we present the TCP/IP, VTAM and VTAM AnyNet definitions we used on our z/OS system.

Basic VTAM definitions

TCP62 requires CICS and VTAM to be correctly configured to accept LU 6.2 parallel sessions. In our VTAM network, our network was already configured as USIBMSC (using NETID=USIBMSC in the VTAM startup options). We also configured a VTAM LOGMODE for the APPCMODE mode group and specified PARSESS=YES on our VTAM APPL definition. For further details on defining VTAM definitions, refer to 2.3, "APPC connections and VTAM" on page 25.

TCP/IP

The TCP/IP address (1) and host name for the z/OS system are defined by default in the PROFILE.TCPIP and TCPIP.DATA data sets. In our configuration the data set was TCPIPMVS.SC66.TCPPARMS and it has members TCPPROF and TCPDATA. The best way to find these data sets is to look from SDSF DA for active started tasks or from your PROCLIB. These members have a lot of information but we are only interested with the TCPIPJOBNAME, HOSTNAME, DOMAINNAMEORIGIN and HOME parameters. The first three can be found from the TCPDATA member, and the HOME parameter is found in the TCPPROF member. In our configuration:

TCPIPJOBNAME	TCPIPMVS is the TCP/IP started task name
HOSTNAME	wtsc66 is the host name of our z/OS system
DOMAINNAMEORIGIN	itso.ibm.com is the domain name
HOME	9.12.6.6 is the TCP/IP address

> **Tip:** You can also use the TSO command **`HOMETEST`** to verify the IP configuration on your z/OS system.

TCP/IP major node

The VTAM TCP/IP major node defines the AnyNet interface between TCP/IP and VTAM. You need only one TCP/IP major node for each LPAR. If you do not know your TCP/IP major node name, you can see if it is defined using the VTAM command `D NET,MAJNODE`. On our system, this displayed the following entry:

```
IST089I APTCP01 TYPE = TCP/IP MAJOR NODE, ACTIV
```

Our TCP/IP major node definition is shown in Example 3-1 on page 45. The important parameters are the port and possibly the TCP/IP job name (TCPIPJOB), if you are using multiple TCP/IP stacks.

Example 3-1 Content of ESA.SYS1.VTAMLST(APTCP01)

```
APTCP01  VBUILD TYPE=TCP,
               CONTIMER=25,
               DGTIMER=40,
               DNSUFX=ITSO.IBM.COM,
               EXTIMER=5,
               IATIMER=60,
               PORT=397,
               TCB=10,
               TCPIPJOB=TCPIPMVS
TCP01GRP       GROUP  ISTATUS=ACTIVE
TCP01LNE       LINE   ISTATUS=ACTIVE
TCP01PU        PU     ISTATUS=ACTIVE
```

Tip: The DNSUFX is the domain name suffix used by VTAM when sending outbound allocates. This does not need to match the AnyNet domain name suffix used in the CICS TG Configuration Tool (Figure 3-4 on page 47), but we set it to the same value for simplicity. In practice, we found this value is only required by VTAM if the CICS region is initiating binds (starting the connection), which is not the case with a TCP62 configuration.

Dynamic LU name generation

We also configured dynamic creation of partner LUs in VTAM. This simplifies the administration required in VTAM. To enable dynamic LUs, we specified DYNLU=YES as a startup option for VTAM. A CDRSC definition is in effect a partner LU 6.2 definition in VTAM for the LU on the workstation. Using dynamic LUs meant that VTAM would create these definitions as required.

Tip: If you do not have DYNLU=YES as a startup option for VTAM, then you will have to statically define cross-domain resource (CDRSC) definitions to VTAM for all your CICS TG TCP62 LUs. This also means you cannot use the dynamic LU name generation function in the CICS TG, since you will be unable to know the names of your LUs in VTAM. In our example the following definitions would suffice:

```
APTCP0C  VBUILD TYPE=CDRSC
CCLI62LU  CDRSC  ALSLIST=TCP01PU
```

For further details on configuring CDRSC definitions for LU 6.2, refer to *AnyNet SNA over TCP/IP,* SC31-8578.

3.4 TCP62 definitions for the CICS TG

We used the CICS TG Configuration Tool to define the settings for TCP62 communication. The Configuration Tool generates the CTG.INI file, which is located in the \bin subdirectory. This file is read by the CICS TG when started and used to establish a connection to a CICS Server. We defined the following parameters as shown in Figure 3-3.

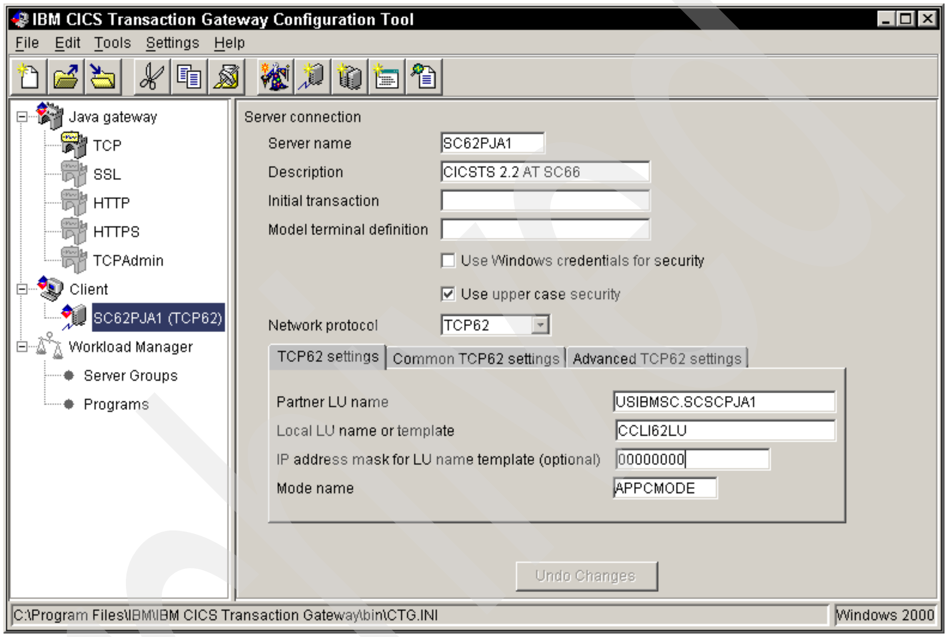


Figure 3-3 TCP62 settings in Configuration Tool

- **Server name**

  An arbitrary name for your CICS server connection. We specified the name SC62PJA1.

- **Description**

  An arbitrary description for the CICS server.

- **Network protocol**

  The protocol for communication with the CICS server (in our example, TCP62).

- **Partner LU name 4.5**

  The fully qualified LU name of the CICS region as it is known in VTAM. This is the APPL ID of the CICS region preceded by the SNA network name (in our example, USIBMSC.SCSCPJA1).

- **Mode name 6**

  The SNA mode name to be used when connecting to the CICS region (in our example, APPCMODE). This must match the mode name in the CICS SESSIONS definition. If you are using CICS connection autoinstall, this is must also be specified in the CICS SESSIONS autoinstall template (CBPS).

- **Local LU name or template**

  The name of the local LU to be used when connecting to the CICS region. Because we were using static LU names, we specified CCLI62LU.

- **IP address mask for LU name template**

  This is the IP address mask used by the CICS TG to dynamically create LU names when using templates; this must be a 32-bit hexadecimal value. Only set this if you are using CICS TG dynamic LU name generation. Since we were using static names, we let this default to 00000000.

Our settings for the Common TCP62 settings tab are shown in Figure 3-4 and explained in the following text.

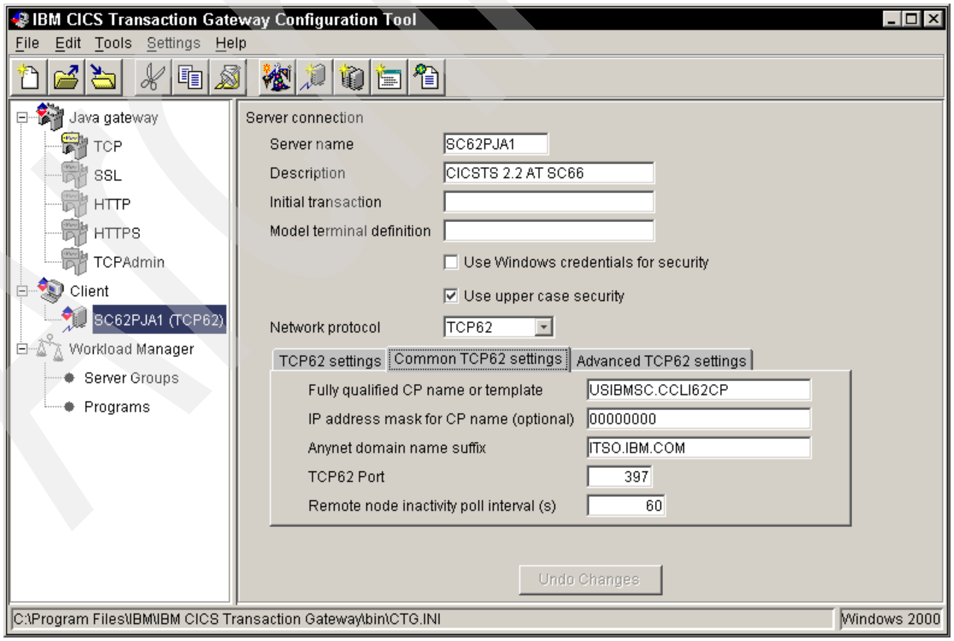


Figure 3-4 Common TCP62 settings in the Configuration Tool

- **Fully qualified CP name or template**

  The fully qualified control point (CP) name for the TCP62 SNA node, or a template. We defined this as USIBMSC.CCLI62CP.

- **IP address mask for CP name**

  An IP address mask for dynamic generation of CP names. We let this default to zeros (00000000), since we did not use this parameter.

- **TCP62 Port 3**

  This is the TCP/IP port used by VTAM to listen for MPTN packets. We specified 397, which is the default AnyNet port, which we also specified in the PORT statement in our VTAM TCP major node (Figure 3-1 on page 45).

> On some UNIX platforms (such as Solaris) if you run the Client daemon as a non-root process you will be unable to use port 397 and will need to specify a non-restricted port number.

TCP62 dynamic LU names

The TCP62 hashing algorithm provides a means of dynamically generating unique LU62 names for the CICS TG based on the local IP address. This provides the benefit that you don't have to administer LU name allocation, and the same CTG.INI file can be used on all your workstations.

To support dynamic LU name generation, you must also have configured the following features of VTAM and CICS:

- VTAM dynamic partner LUs (DYNLU=YES)
- CICS APPC CONNECTION autoinstall

As well as using static LU names, we also tested our connection to CICS using dynamic LU names. To enable this feature, you should set the following parameters in the CICS TG Configuration Tool TCP62 settings (see Figure 3-3 on page 46).

- IP address mask for LU name template
- Local LU template

> **Tip:** Contrary to the CICS TG documentation, we found that we did not need to configure the CP name template or IP address mask when using dynamic LU names. This is because in SNA networking terminology, the AnyNet LU 6.2/IP connection (link station) from the CICS TG to VTAM is a low-entry node (LEN) link. Therefore, the CP name is only used locally (on the CICS TG machine), so it does not need to be defined to VTAM and does not need to be unique in the network.

The IP address mask should be set to the hexadecimal version of your IP subnet mask. This will ensure that the LU name generated is unique within your local IP subnet. We used FFFFFE00, the hexadecimal value of our TCP/IP subnet mask 255.255.254.0. To determine your IP address and TCP/IP subnet mask, you can use the Windows **ipconfig /all** command as shown in Example 3-2.

Example 3-2 ipconfig utility

```
C:> ipconfig /all

Windows 2000 IP Configuration

      Host Name . . . . . . . . . . . . : volga
      Primary DNS Suffix  . . . . . . . : almaden.ibm.com
        Node Type . . . . . . . . . . . . : Hybrid
        IP Routing Enabled. . . . . . . . : No
        WINS Proxy Enabled. . . . . . . . : No
      DNS Suffix Search List. . . . . . : itso.ibm.com
                                          almaden.ibm.com

Ethernet adapter Local Area Connection 3:

     Connection-specific DNS Suffix  . :
     Description . . . . . . . . . . . : Intel(R) PRO/100 VE Desktop Connection
        Physical Address. . . . . . . . . : 00-02-55-4A-7A-67
        DHCP Enabled. . . . . . . . . . . : No
        IP Address. . . . . . . . . . . . : 9.1.38.39
        Subnet Mask . . . . . . . . . . . : 255.255.254.0
        Default Gateway . . . . . . . . . : 9.1.38.1
        DNS Servers . . . . . . . . . . . : 9.1.8.254
                                            9.14.1.30
```

IP subnets: The subnet is the mask that is applied to an IP address to determine which bits in an IP address are to be used for local addresses (that is, addresses on the local subnet) and which bits are to be used for the network address (that is, on the Internet). Thus, with our subnet mask of 255.255.254.0 (or in binary 11111111.11111111.11111110.00000000) we have 8 bits free in the fourth byte (0) and 1 bit free in the third byte (254), giving a total of 9 free bits. This means that we have 2^9 or 512 unique network addresses available in the subnet.

For each substitution character the TCP62 hashing algorithm uses 5 bits from the IP address. The CICS connection autoinstall program (DFHZATDY) uses the last four characters of the LU name to generate the connection name, which has to be unique within the CICS region. Therefore, if you wish to ensure that the dynamically generated TCP62 LU names and the CICS connections names are unique, you should obey the following rules:

- Set the TCP62 IP address mask to be equal to the TCP/IP subnet mask.
- Set the TCP62 LU name template to have four or less trailing * characters.
- Ensure you use enough substitution characters (*) to accommodate all the unique IP addresses in your subnet as shown in Table 3-3.

> **Generating unique LU names:** There are 32 different values that the algorithm can use for each substitution character. Generating 32 different values requires 5 unique bits (2^5=32). This means that up to 5 bits are used for each substitution character, starting from the right. Based upon this fact, Table 3-3 helps you determine the minimum number of substitution characters to use in your template name to guarantee generated LU names will be unique within your subnet.
>
> In our example, we have a subnet mask of 255.255.254.0, which has 9 bits free for local network addresses, and therefore a possible 2^9 (512) unique IP address in the subnet. We used two substitution characters that can accommodate up to 2^{10} combinations (1024), which is more than enough to cover the 512 unique IP addresses in our subnet.

Table 3-3 Values required to generate unique LU names

Number of free bits in the IP address mask	Number of substitution characters you should allow	Maximum number of unique names this gives you
16-20	4 (XXXX****)	2^{20} = 1,048,576
11-15	3 (XXXXX***)	2^{15} = 32,768
6-10	2 (XXXXXX**)	2^{10} = 1024
1-5	1 (XXXXXXX*)	2^5 = 32

In our example we used the following values.

LU name template	CCLIEE**
IP address	9.1.38.39
IP subnet mask	FFFFFE00
Number of bits for local addresses	9

Using these values, the TCP62 algorithm created a local LU name of CCLIEE17 and the CICS connection autoinstall program, DFHZATDY, created a connection called EE17. We can be sure that this name is unique for all 512 different network addresses in our TCP/IP subnet.

> **Note:** Although using this technique ensures our LU names will be unique within the subnet, it is possible that two machines from different subnets will connect to the same CICS region, causing a potential name clash. In this case, we suggest that you ensure that you only use three (or less) substitution characters and that the fifth characters in the LU name template is different for each subnet.

To help you in determining what a given TCP62 LU name will be for different inputs, we developed a sample utility, tcp62localluexe, that takes a given IP address, subnet mask, and LU name template, and generates the LU name that TCP62 will use. This use of this utility is illustrated in Figure 3-3 and it is provided with the additional material for this book.

Example 3-3 tcp62locallu utility

```
C:> tcp62locallu.exe CCLIE*** FFFFFE00 9.1.38.39
Local LU template: CCLIE***
Address mask     : FFFFFE00
IP address       : 9.1.38.39
Local LU is CCLIEE17
```

Configuring TCP/IP host names

Since TCP62 flows SNA LU 6.2 packets over TCP/IP, you need to define a mapping of the SNA partner LU name to an IP address. Our partner LU is our CICS region, with a fully qualified LU name of USIBMSC.SCSCPJA1. This needs to be mapped to the IP address of our z/OS system wtsc66, which is 9.12.6.6.

The function in TCP62 achieves this by generating a host name from the concatenation of the following elements separated by periods:

LU name (5) + network name of the LU (4) + domain name suffix (2)

We defined a mapping of this host name to the IP address of our z/OS system using the TCP/IP hosts file (C:\WINNT\system32\drivers\etc\hosts) on our Windows 2000 workstation. The following is the line we added to our hosts file.

```
9.12.6.6 SCSCPJA1.USIBMSC.ITSO.IBM.COM
```

If you wish to use the same CTG.INI configuration file across multiple client workstations, you will also need to provide a means of resolving the TCP62 host name to the correct z/OS IP address on all workstations. Instead of just using the TCP/IP hosts file, it is possible to add an entry to the Domain Name System (DNS) server used by the workstations.

However, if you do use a DNS entry to resolve the TCP62 host name, you will most likely need to create a new subdomain since the SNA network name (USIBMSC) is prefixed onto the domain name suffix to form a new domain (USIBMSC.ITSO.IBM.COM) in which the LU name (SCSCPJA1) is located.

Tip: If you are using the hosts file to resolve the z/OS host name, there is no reason why the domain name suffix should match the TCP/IP network name (itso.ibm.com). However, we used this to simplify our configuration.

Testing the configuration

After we installed and configured the software components, we tested our configuration as follows:

1. Pinged the TCP/IP connection to the z/OS system using our AnyNet host name SCSCPJA1.USIBMSC.ITSO.IBM.COM as shown in Example 3-4.

Example 3-4 Ping to check host name

```
C:\>ping SCSCPJA1.USIBMSC.ITSO.IBM.COM

Pinging SCSCPJA1.USIBMSC.ITSO.IBM.COM [9.12.6.6] with 32 bytes of data:

Reply from 9.12.6.6: bytes=32 time=110ms TTL=52
Reply from 9.12.6.6: bytes=32 time=120ms TTL=52
Reply from 9.12.6.6: bytes=32 time=120ms TTL=52
Reply from 9.12.6.6: bytes=32 time=120ms TTL=52
```

2. Checked that the port number 397 is listening at our host.

   We used a shareware product called ScanPort to scan the ports in use on our z/OS system. The output from our scan is shown in Figure 3-5 on page 53. Using ScanPort, we can see that port 397 is in use on our host at address 9.12.6.6. You may note that the Detail field tells us that this is registered as the MPTN port on our local machine. Multi-Protocol Transport Network (MPTN) is the network transport used by TCP62 and AnyNet to send LU 6.2 packets over an IP network.

   If you wish to use the ScanPort utility, it can be downloaded from the following Web site:

   `http://www.dataset.fr/eng/scanport.html`

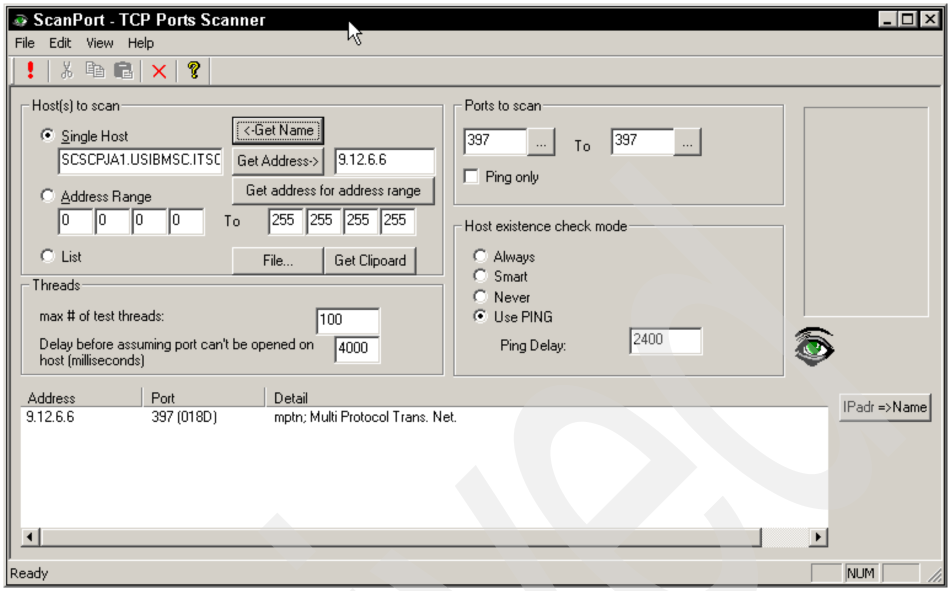


Figure 3-5 ScanPort utility

3. Started the CICS region SCSCPJA1.
4. Started the CICS TG connection to the CICS region using the command `CICSCLI /S=SC62PJA1`. (SC62PJA1 is the server name given in the Client configuration step in Figure 3-3 on page 46.)
5. Checked the status of the connection to the CICS region using the command `CICSCLI /L`. The connection status is available, as shown in Example 3-5.

Example 3-5 Checking connection status

```
C:\>CICSCLI /L
CCL8001I CICSCLI - CICS Client Control Program
CCL0002I (C) Copyright IBM Corporation 1994,2001. All rights reserved.
CCL8041I The CICS Client is using the following servers:
CCL8042I Server 'SC62PJA1' (using 'TCP62' to 'USIBMSC.SCSCPJA1') is available
```

6. Issued the `CICSTERM /S=SC62PJA1` command to install a client terminal on the CICS region.
7. Ran the CICS-supplied transaction `CEMT I CONN` to display the status of our TCP62 connection (Figure 3-6 on page 54). You can see that our TCP62 connection has been autoinstalled as 62LU, which are the last four characters of the LU name (CCLI62LU), and is marked Acq, meaning SNA sessions are bound. The other connection displayed is CBPS, the APPC connection autoinstall template.

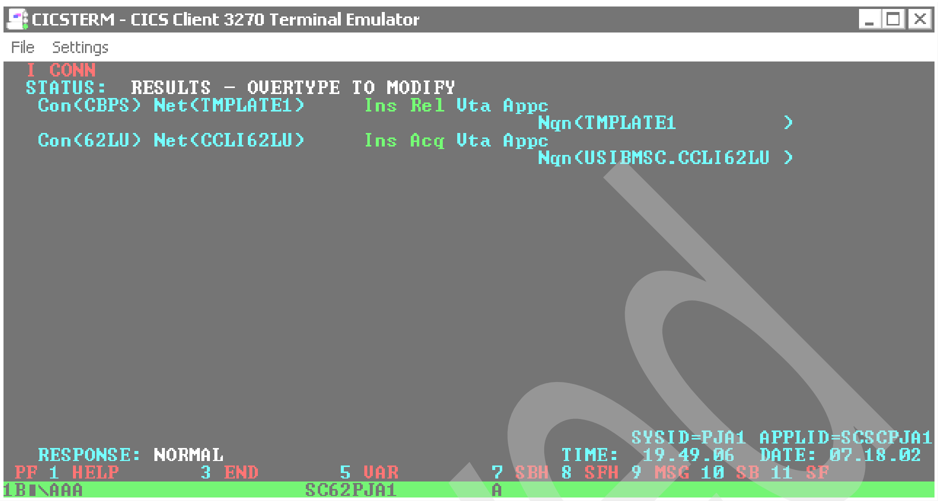


Figure 3-6 CEMT I CONN

Lastly we checked the CICS region CSMT log to view the messages from the CICS connection autoinstall, as shown in Example 3-6.

Example 3-6 CICS log messages from the connection autoinstall

```
DFHZC6935 I 05/28/2002 13:58:50 SCSCPJA1 Autoinstall for connection 62LU with netname CCLI62LU
using model or template CBPS successful.
DFHZC3461 I 05/28/2002 13:58:50 SCSCPJA1 -AAY CSNE Node CCLI62LU session started.  ((1) Module
name: DFHZOPX) NQNAME   -AAY,CSNE,13:58:50,USIBMSC  CCLI62LU
DFHZC4900 I 05/28/2002 13:58:50 SCSCPJA1 -AAY CLS1 CNOS received from Node CCLI62LU System 62LU
Modename APPCMODE, Max =8, Win=7, successful. ((1) Module name: DFHZGCN)
DFHZC3461 I 05/28/2002 13:58:50 SCSCPJA1 -AAX CSNE Node CCLI62LU session started.  ((2) Module
name: DFHZOPX) NQNAME   -AAX,CSNE,13:58:50,USIBMSC  CCLI62LU
DFHZC3461 I 05/28/2002 13:58:50 SCSCPJA1 -AA6 CSNE Node CCLI62LU session started.  ((1) Module
name: DFHZOPX) NQNAME   -AA6,CSNE,13:58:50,USIBMSC  CCLI62LU
DFHZC5966 I 05/28/2002 16:10:46 SCSCPJA1 INSTALL started for TERMINAL (  \AAA) (Module name:
DFHBSTZ).
```

3.5 Problem determination

In this section, we document information we learned while configuring this scenario, as well as further information on problem determination and tracing.

3.5.1 Tips and utilities

In this section, you will find useful commands and utilities for debugging any problems with your configuration. You will also find some additional information about topics discussed in this chapter.

Logs

For problem determination at the client side, it is best to check the CICSCLI.LOG under the /bin directory for error messages. If this does not give enough information, you should start the CICS TG connection with trace as shown in 3.5.2, “Tracing” on page 61.

For problem determination with APPC connections into CICS TS, refer to the following logs:

- z/OS system log (SDSF.LOG) for VTAM information messages
- CICS console and CSMT logs for CICS messages

Windows TCP/IP status

To check the status of TCP/IP as used by AnyNet on the workstation, you can issue the **netstat /a /n** command to view socket usage as shown in Example 3-7.

Example 3-7 Client side netstat information

```
C:\>netstat /a /n
Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:397            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1025           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1039           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1073           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1161           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1186           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1187           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:9495           0.0.0.0:0              LISTENING
  TCP    9.1.38.39:139          0.0.0.0:0              LISTENING
  TCP    9.1.38.39:397          9.12.6.6:1263          ESTABLISHED
  TCP    9.1.38.39:1033         0.0.0.0:0              LISTENING
  TCP    9.1.38.39:1072         0.0.0.0:0              LISTENING
```

```
  TCP    9.1.38.39:1072         9.1.38.11:139          ESTABLISHED
  TCP    9.1.38.39:1073         9.12.14.8:23           ESTABLISHED
  TCP    9.1.38.39:1161         9.12.6.6:23            ESTABLISHED
  TCP    9.1.38.39:1186         9.12.6.6:397           ESTABLISHED
  TCP    9.1.38.39:1187         9.12.6.6:397           ESTABLISHED
  UDP    0.0.0.0:135            *:*
  UDP    0.0.0.0:397            *:*
  UDP    0.0.0.0:445            *:*
  UDP    0.0.0.0:1036           *:*
  UDP    0.0.0.0:1185           *:*
  UDP    9.1.38.39:137          *:*
  UDP    9.1.38.39:138          *:*
  UDP    9.1.38.39:500          *:*
```

This shows that the CICS TG TCP62 function is using the TCP and UDP ports 397 on our workstation (9.1.38.39), and that three sessions are established using TCP connections with the z/OS system (9.12.6.6).

> **UDP:** Note that AnyNet uses UDP connections for LU 6.2 expedited flows, as well as for session unbinds and for the AnyNet inactivity timers. Therefore if you use a firewall to block certain ports, you must open both the TCP *and* UDP ports. If you do not open the UDP ports, you will see sessions disconnected with inactivity, and failures during session unbinds.

Display MVS TCP/IP status

The `D TCPIP` command issued from SDSF displays the status of TCP/IP on the z/OS system as shown in Example 3-8.

Example 3-8 Status of TCP/IP

```
D TCPIP
RESPONSE=SC66
 EZAOP50I TCPIP STATUS REPORT 014
 COUNT   TCPIP NAME   VERSION    STATUS
 -----   ----------   --------   ------------------------
     1   TCPIPOE      CS V1R2    ACTIVE
     2   TCPIPMVS     CS V1R2    ACTIVE
 *** END TCPIP STATUS REPORT ***
 EZAOP41I 'DISPLAY TCPIP' COMMAND COMPLETED SUCCESSFULLY
```

In our example, we were using two TCP/IP stacks on our z/OS system, TCPIPMVS and TCPIPOE. We used the TCPIPMVS stack with the VTAM AnyNet feature.

To display the TCP/IP sockets in use by AnyNet, you can use the MVS `NETSTAT` command as shown in Example 3-9 on page 57. This will give you the same information that you got from the client side. The following screen shows the NETSTAT output with sessions bound from CICS to our TCP62 client on IP address 9.1.38.39. As you can see, AnyNet uses both TCP and UDP protocols.

Example 3-9 MVS NETSTAT

```
MVS TCP/IP NETSTAT CS V1R2       TCPIP NAME: TCPIPMVS         01:02:47
User Id  Conn     Local Socket           Foreign Socket         State
-------  ----     ------------           --------------         -----
VTAMBUDI 0000A26F 9.12.6.6..397          9.1.38.39..1186        Establsh
VTAMBUDI 0000A271 9.12.6.6..397          9.1.38.39..1187        Establsh
VTAMBUDI 00001E57 0.0.0.0..397           0.0.0.0..0             Listen
VTAMBUDI 0000A270 9.12.6.6..1263         9.1.38.39..397         Establsh
VTAMBUDI 00001E56 0.0.0.0..397           *..*                   UDP
VTAMBUDI 00001E5D 9.12.6.6..2978         *..*                   UDP
```

Display VTAM AnyNet status

The VTAM command `D NET,STATS,TYPE=VTAM` shows whether VTAM AnyNet is installed and the number of TCP major nodes defined to VTAM as shown in Example 3-10.

Example 3-10 AnyNet part of the VTAM statistics

```
D NET,STATS,TYPE=VTAM
IST097I DISPLAY ACCEPTED
IST350I DISPLAY TYPE = STATS,TYPE=VTAM 225
IST1349I COMPONENT ID IS 5695-11701-120
IST1345I    ID      VALUE        DESCRIPTION
IST1227I   130        YES = ANYNET/MVS SNA OVER TCP/IP INSTALLED
IST1227I   127          1 = TCP/IP MAJOR NODES
IST1227I   128         10 = MAXIMUM TCB VALUE FOR TCP/IP MAJOR NODES
IST1227I   129          3 = TCP/IP LU-LU SESSIONS
```

The VTAM command `D NET,ID=APTCP01` displays the status of the TCP major node as shown in Example 3-11. If you do not know the VTAM major node name, you can use the command `D NET,MAJNODES` to list all the defined major nodes.

Example 3-11 TCP major node

```
D NET,ID=APTCP01
IST097I DISPLAY ACCEPTED
IST075I NAME = APTCP01, TYPE = TCP/IP MAJOR NODE 236
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
IST1342I DNSUFX = ITSO.IBM.COM
IST1692I TCB = 10  TCP PORT =   397
IST1400I DGTIMER  =    40  EXTIMER =     5
IST1406I CONTIMER =    25  IATIMER =    60
IST654I I/O TRACE = OFF, BUFFER TRACE = OFF
```

```
IST314I END
```

The VTAM command `D NET,ID=TCP01PU,E` displays the status of a TCP physical unit (PU) in VTAM, as shown in Example 3-12. TCP01PU is the TCP PU defined in the TCP major node name. The important information from this command is that AnyNet TCP PU is active. If this is not active, TCP62 clients will not work. Also note that dynamic LUs are supported, so we can use dynamic LU names in VTAM.

Example 3-12 Status of TCP PU

```
D NET,ID=TCP01PU,E
IST097I DISPLAY ACCEPTED
IST075I NAME = TCP01PU, TYPE = PU_T2.1 241
IST486I STATUS= ACTIV--L--, DESIRED STATE= ACTIV
IST1043I CP NAME = ***NA***, CP NETID = USIBMSC, DYNAMIC LU = YES
IST1589I XNETALS = NO
IST081I LINE NAME = TCP01LNE, LINE GROUP = TCP01GRP, MAJNOD = APTCP01
IST654I I/O TRACE = OFF, BUFFER TRACE = OFF
IST1500I STATE TRACE = OFF
IST355I LOGICAL UNITS:
IST080I CCLI62LU ACT/S----Y
IST314I END
```

Display VTAM LU status

The VTAM command `D NET,ID=SCSCPJA1,E` displays the status of the VTAM LU for the CICS region and any sessions bound as shown in Example 3-13.

Example 3-13 VTAM information from our CICS region SCSCPJA1

```
D NET,ID=SCSCPJA1,E
IST097I DISPLAY ACCEPTED
IST075I NAME = USIBMSC.SCSCPJA1, TYPE = APPL 358
IST486I STATUS= ACT/S, DESIRED STATE= ACTIV
IST1447I REGISTRATION TYPE = CDSERVR
IST977I MDLTAB=***NA*** ASLTAB=***NA***
IST861I MODETAB=MTAPPC USSTAB=***NA*** LOGTAB=***NA***
IST934I DLOGMOD=***NA*** USS LANGTAB=***NA***
IST1632I VPACING =  0
IST597I CAPABILITY-PLU ENABLED  ,SLU ENABLED  ,SESSION LIMIT NONE
IST231I APPL MAJOR NODE = APCPJA1
IST654I I/O TRACE = OFF, BUFFER TRACE = OFF
IST1500I STATE TRACE = OFF
IST271I JOBNAME = SCSCPJA1, STEPNAME = SCSCPJA1, DSPNAME = IST99C78
IST1050I MAXIMUM COMPRESSION LEVEL - INPUT = 0, OUTPUT = 0
IST1633I ASRCVLM = 1000000
IST1634I DATA SPACE USAGE: CURRENT = 0 MAXIMUM = 512
```

```
IST171I ACTIVE SESSIONS = 0000000003, SESSION REQUESTS = 0000000000
IST206I SESSIONS:
IST634I NAME     STATUS         SID              SEND RECV VR TP NETID
IST635I CCLI62LU ACTIV/SV-S ECF7040A341DB863 0001 0000       USIBMSC
IST635I CCLI62LU ACTIV-P    101512001C04D207 0001 0001       USIBMSC
IST635I CCLI62LU ACTIV/SV-P 101511001C04D207 0001 0001       USIBMSC
IST314I END
```

This screen provides a wealth of information, but the most important information is that the LU status is active with sessions (ACT/S). It is using the mode table MTAPPC (which contains our log mode APPCMODE), and it has three active sessions. To obtain more information on the active sessions listed, you can use the VTAM command `D NET,SESSIONS,LU1=CCLI62LU,LIST=ALL` as shown in Example 3-14:

Example 3-14 CICS TG APPC sessions

```
D NET,SESSIONS,LU1=CCLI62LU,LIST=ALL
IST097I DISPLAY ACCEPTED
IST350I DISPLAY TYPE = SESSIONS 361
IST873I        PLU              SLU                SID        STATUS
IST874I USIBMSC.SCSCPJA1  USIBMSC.CCLI62LU  ECF7040A341DB863 ACTIV/SV
IST874I USIBMSC.CCLI62LU  USIBMSC.SCSCPJA1  101512001C04D207 ACTIV
IST874I USIBMSC.CCLI62LU  USIBMSC.SCSCPJA1  101511001C04D207 ACTIV/SV
IST924I -------------------------------------------------------------
IST878I NUMBER OF PENDING SESSIONS =         0
IST924I -------------------------------------------------------------
IST878I NUMBER OF ACTIVE  SESSIONS =         3
IST924I -------------------------------------------------------------
IST878I NUMBER OF QUEUED  SESSIONS =         0
IST924I -------------------------------------------------------------
IST878I NUMBER OF TOTAL   SESSIONS =         3
IST314I END
```

You may also use the following command, as shown in Example 3-15, where SID is a session ID:

```
D NET,SESSIONS,SID=101512001C04D207
```

If you display all the three sessions, you will find out that the first one is for SNASVCMG (the LU 6.2 service manager mode) and the last two are the real CICS TG sessions for the APPCMODE.

Example 3-15 CICS TG session information

```
D NET,SESSIONS,SID=101512001C04D207
IST097I DISPLAY ACCEPTED
IST350I DISPLAY TYPE = SESSIONS 367
```

```
IST879I PLU/OLU REAL = USIBMSC.CCLI62LU  ALIAS = ***NA***
IST879I SLU/DLU REAL = USIBMSC.SCSCPJA1  ALIAS = ***NA***
IST880I SETUP STATUS = ACTIV
IST875I ALSNAME TOWARDS PLU = TCP01PU
IST933I LOGMODE=APPCMODE, COS=*BLANK*
IST1165I LOCAL TCP/IP ADDRESS = 9.12.6.6..397
IST1165I REMOTE TCP/IP ADDRESS = 9.1.38.39..1187
IST1635I PLU HSCB TYPE: BSB LOCATED AT ADDRESS X'0E701238'
IST1635I SLU HSCB TYPE: FMCB LOCATED AT ADDRESS X'0874A218'
IST1636I PACING STAGE(S) AND VALUES:
IST1644I PLU--STAGE 1-----|-----STAGE 2--SLU
IST1638I STAGE1: PRIMARY TO SECONDARY DIRECTION - FIXED
IST1640I         SECONDARY RECEIVE      = *NA*
IST1641I STAGE1: SECONDARY TO PRIMARY DIRECTION - FIXED
IST1642I         SECONDARY SEND: CURRENT =     0     NEXT =     0
IST1638I STAGE2: PRIMARY TO SECONDARY DIRECTION - ADAPTIVE
IST1639I         PRIMARY SEND: CURRENT   =     0     NEXT = 32767
IST1640I         SECONDARY RECEIVE      = 32767
IST1641I STAGE2: SECONDARY TO PRIMARY DIRECTION - ADAPTIVE
IST1642I         SECONDARY SEND: CURRENT =     0     NEXT =     7
IST1643I         PRIMARY RECEIVE        =     0
IST1714I NO PATH INFORMATION EXISTS
IST314I END
```

SNA sense codes

We found the Personal Communications GetSense utility helpful for quickly looking up SNA sense code. To launch GetSense from Windows, click **Start -> Programs -> IBM Personal Communications -> Administrative and PD Aids** and click **Display SNA Sense Data**. The GetSense window will appear, as shown in Figure 3-7.

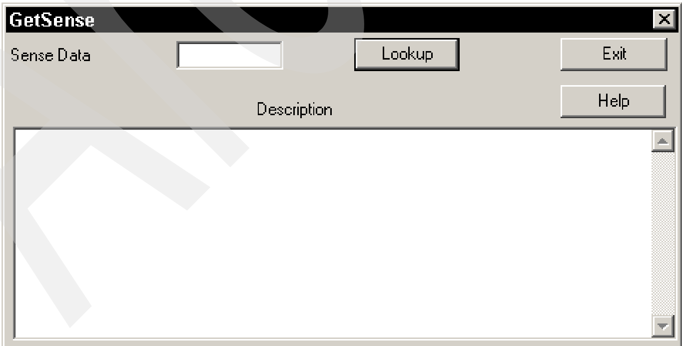


Figure 3-7 GetSense window

SNA sense data can be entered in the Sense Data field and a detailed sense code description displayed by clicking **Lookup**.

3.5.2 Tracing

In this section, we provide information on how to use the trace facilities provided with the Client daemon to debug a simple TCP62 communications problem.

Client daemon

The first place to look for error messages when using the CICS TG on the Windows system is the log file. This is specified in the Client configuration parameters in the CICS TG Configuration Tool. We used the default name CICSCLI.LOG, which can be found in the CICS TG \bin subdirectory.

The first step to useful tracing with the Client daemon is identifying which components should be traced. A full list is shown in Example 3-16 and can be viewed using the command `CICSCLI /M`. If in doubt, you should trace with the default selected components, since these have been selected to be sufficient to diagnose most problems. The default components have an `X` in front of them in the following example. To turn on all the parameters, you can use the `/M=ALL` option.

Example 3-16 CICS TG client trace options

```
C:\>cicscli /m
CCL8001I CICSCLI - CICS Client Control Program
CCL0002I (C) Copyright IBM Corporation 1994,2001.  All rights reserved.
CCL1120I Trace has not been started
CCL1100I The following list shows what components can be traced
CCL1101I An 'X' indicates which components are enabled
CCL1102I       CICSCLI Command Process     [CLI]
CCL1103I       Interprocess Communication  [TRN]
CCL1104I    X  Protocol Drivers (Level 1)  [DRV.1]
CCL1105I       Protocol Drivers (Level 2)  [DRV.2]
CCL1106I    X  API (Level 1)               [API.1]
CCL1107I       API (Level 2)               [API.2]
CCL1108I    X  Client Daemon               [CCL]
CCL1109I       Terminal Emulators          [EMU]
CCL1111I       C++ Class Libraries         [CPP]
CCL1112I       Workload Manager            [LMG]
CCL1113I       CICS Client Service for NT  [SER]
```

Trace is written to a binary file called, by default, CICSCLI.BIN. To read the trace, you must format the binary file into a text file by using the `CICSFTRC /D` command. This will produce a viewable file called CICSCLI.TRC. Both CICSCLI.BIN and CICSCLI.TRC are found, by default, in the \BIN subdirectory.

The CICS TG Client daemon trace is started with the `CICSCLI /D` command. If you need to trace the client from startup, you can specify all the parameters needed together on the command line, such as:

```
CICSCLI /S=SCSCPJA1 /D=9999 /M=ALL
```

For more information on Client daemon tracing, wrapping trace, and formatting options, refer to *CICS Transaction Gateway Windows Client Administration,* SC34-5940.

Example trace

To provide an example tracing scenario we disabled connection autoinstall on our CICS region and tried to connect using TCP62 from our Windows workstation. To disable autoinstall on CICS, we used the CICS command:

```
CEMT SET AUTOI PROGR(DFHZATDX)
```

To start the Client daemon and format the trace, we did the following from a Windows command prompt:

1. We started the CICS TG connection to the CICS region using the command:

   ```
   CICSCLI /S=SC62PJA1 /D=9999 /M=DRV.1,DRV.2
   ```

2. We stopped the CICS TG connection to the CICS region using the command:

   ```
   CICSCLI /X
   ```

3. We formatted the trace with detailed formatting using the command:

   ```
   CICSFTRC /D
   ```

The trace was output to the file CICSCLI.TRC in the CICS TG bin directory. As shown in Example 3-17, the first network flow is an MPTN_Connect, from the Client daemon with the node address USIBMSC.CCLI62LU. The destination address is our CICS server, USIBMSC.SCSCPJA1.

Example 3-17 trace of connect

```
12:23:20.040 [00000780,000007f0] [5874] DRV:CCL5899 TCP62: connected
006c-SC62PJA1
12:23:20.050 [00000780,000007f0] [5879] DRV:CCL5894 TCP62: send ctl
006c-SC62PJA1 Length=254
...
________________________________________________MPTN Flow - Start_
00000004  Command Type = (0x80) MPTN_Connect
...
          Destination User Address:
00000009    MPTN Qualifier = (0x0b) SNA network-qualified LU name
0000000a    Address Mode = (0x01) individual address
0000000c    Node Address = "USIBMSC.SCSCPJA1"
0000001c    Local Address = (0x01) null
```

```
         Source User Address:
0000001d    MPTN Qualifier = (0x0b) SNA network-qualified LU name
0000001e    Address Mode = (0x01) individual address
00000020    Node Address = "USIBMSC.CCLI62LU"
00000030    Local Address = (0x01) null
```

The response from our CICS server is shown in Example 3-18. The MPTN flow details show that the request was recognized by the destination but has been rejected. The SNA flow details show that the destination has responded with an UNBIND request for the Client daemon, identified by the fully qualified CP name USIBMSC.CCLI62CP. The SNA sense data is 0x08100000.

Example 3-18 Trace of CICS response

```
12:23:20.170 [00000780,000007f0] [5863] DRV:CCL5898 TCP62: Sockets thread
posted, Rc= 1
12:23:20.170 [00000780,000007f0] [5887] DRV:CCL5894 TCP62: recv ctl
006c-SC62PJA1 Length=208
...
________________________________________________MPTN Flow - Start_
00000004  Command Type = (0x80) MPTN_Connect
00000005  Processing Specification:
00000005    (0b111) -ve response, recognised by destination but rejected
00000005    (0b0) Gateway recognised command
00000005    (0b0) Destination recognised command
...
- - - - - - - - - - - - - - - - - - - - - - - - SNA Flow - Start-
...
00000056    Request Code = (0x32) UNBIND
...
00000067      Network-Qualified CP Name = "USIBMSC.CCLI62CP"
            Control Vector:
00000077      Key = (0x35) Extended Sense Data
00000079      Sense Data = 0x08010000
...
12:23:20.170 [00000780,000007f0] [5894] DRV:CCL5815 TCP62: Session
006c-SC62PJA1 MPTNCONN rejected, Rc=0007,0000, sense data=08010000
```

The MPTNCONN rejected message is output further down the trace, giving the SNA sense data. The GetSense utility can be used to determine what this data means, as shown in Figure 3-8 on page 64. The code indicates that the CICS LU is not available.

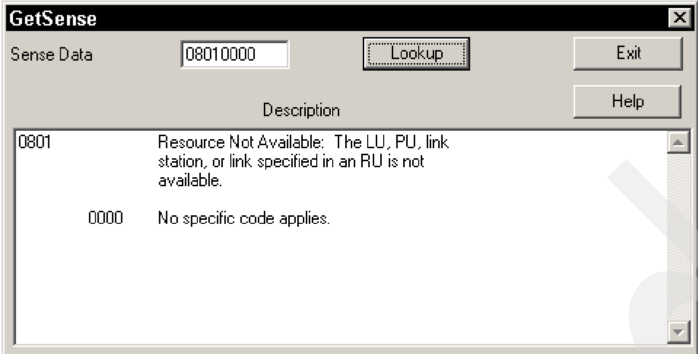


Figure 3-8   GetSense showing the SNA sense data

Finally, we examined the CICS log to find the message DFHZC6921W. As shown in Example 3-19, autoinstall was started for the Client daemon connection, but because we disabled connection autoinstall it was disallowed and therefore the connection was rejected.

Example 3-19   CICS log showing autoinstall rejected

```
DFHZC6907 I 07/09/2002 15:24:07 SCSCPJA1 Autoinstall starting for netname
CCLI62LU. Network qualified name is USIBMSC.CCLI62LU.
DFHZC6921 W 07/09/2002 15:24:07 SCSCPJA1 Autoinstall for NETNAME CCLI62LU has
been disallowed by the autoinstall control program. Code X'FA07'
DFHZC2411 E 07/09/2002 15:24:07 SCSCPJA1 DUMY CSNE CCLI62LU attempted invalid
logon.  ((7) Module name: DFHZATA) NQNAME   DUMY,CSNE,15:24:07,USIBMSC
CCLI62LU
```

Using the CICS messages and codes transaction, CMAC we used the code FA07 for the message DFHZC6921 to determine why connection autoinstall had failed. The output indicates that APPC connection autoinstall is not supported or failed.

Example 3-20   CMAC output for ZC6921

```
X'FA07'   If APPC autoinstall is not supported, use the
          netname to determine which device is
          attempting autoinstall.

          If APPC autoinstall is supported, examine the
          autoinstall control program to determine why it
          has not set the return code to allow the install.
```

4

EXCI connections to CICS

In this chapter, we discuss the implementation and problem diagnosis of the External CICS Interface (EXCI) protocol as used by the CICS TG in the z/OS environment.

4.1 Introduction to EXCI

The EXCI provides a means for non-CICS address spaces on z/OS to link to programs in a CICS region. For example, the EXCI allows batch jobs to connect to CICS and also allows the CICS Transaction Gateway for z/OS to make calls to CICS (Figure 4-1).

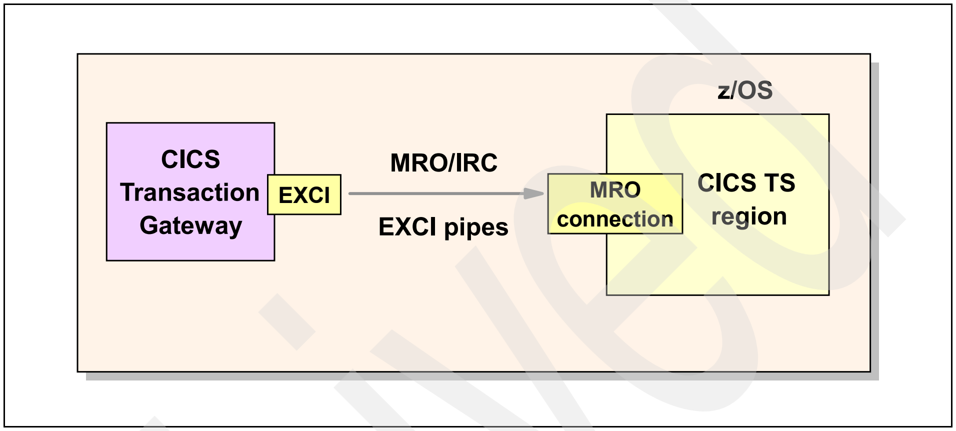


Figure 4-1 CICS client-server connections

On z/OS the use of the EXCI limits the CICS TG to the subset of the ECI that the EXCI supports, and prevents use of the EPI to invoke 3270-based transactions, or the ESI to invoke password expiration management (PEM) functions in CICS. For further details on the different interfaces (ECI, EPI, ESI) provided by the CICS TG, refer to 1.2, “CICS TG: Interfaces” on page 11.

For all connected EXCI client systems, the CICS server region requires both a CONNECTION and SESSIONS definition. The CONNECTION definition identifies the remote system, and one or more SESSIONS definitions are associated with this CONNECTION to define the properties of the sessions.

In the following sections, we provide further information on setting up EXCI on z/OS. For more details, refer to the *CICS External Interfaces Guide,* SC33-1944.

The EXCI protocol is inextricably tied to the CICS Transaction Gateway for z/OS and must be set up in tandem with the CICS TG for z/OS. For details on how we configured the CICS Transaction Gateway for z/OS, refer to Chapter 7, “TCP connections to the Gateway daemon on z/OS” on page 133.

For details on securing connections using the CICS Transaction Gateway, refer to Chapter 6, “CICS TG security scenarios” on page 99.

4.1.1 Software checklist

We used the following levels of software:

- z/OS V1.2
- CICS Transaction Server V2.2

4.1.2 Definitions checklist

The definitions we used to configure this scenario are summarized in Table 4-1. Before you configure the products, we recommend that you decide on definitions for each parameter listed. The Example column shows the values we used in our configuration.

Table 4-1 Definitions checklist for EXCI

CICS region	CICS TG	Example
NETNAME	DFHJVPIPE	SCSCTG5
CONNECTION name		CTG5
RECEIVEPFX		C5

In addition, we also had to deploy our CICS COBOL application EC01, and to configure a DFHCNV data conversion template in CICS for use by this program. Refer to Appendix A, “DFHCNV and CICS data conversion” on page 329 for more details.

4.2 EXCI connections

When creating EXCI connections, the following definitions are required:

- SIT parameters, ISC=YES, IRCSTRT=YES
- CONNECTION definition
- SESSIONS definition

In addition you may need to do the following:

- Add surrogate security profiles
- Modify the EXCI options table DFHXCOPT
- Modify the EXCI user-replaceable module DFHXCURM

4.2.1 EXCI CONNECTION definitions

Figure 4-2 shows the EXCI connection definition we used for the connection to our CICS TG region, SCSCTG5.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0530
 CEDA  ALter Connection( CTG5 )
 Connection     : CTG5
 Group          : CTG5
  DEscription  ==>
CONNECTION IDENTIFIERS
  Netname      ==> SCSCTG5
  INDsys       ==>
CONNECTION PROPERTIES
  ACcessmethod ==> IRc                 Vtam | IRc  | INdirect | Xm
  PRotocol     ==> Exci                Appc | Lu61 | Exci
  Conntype     ==> Specific            Generic | Specific
  SInglesess   ==> No                  No | Yes
SECURITY
 SEcurityname ==>
 ATtachsec    ==> Local               Local | Identify | Verify
                                            | Persistent | Mixidpe

                                             SYSID=PAA6 APPLID=SCSCPAA6
```

Figure 4-2 EXCI connection definition

The important parameters in an EXCI connection definition are:

- **CONNECTION**

  This is the name of the connection itself, as it is known to CICS. Any four-character name can be used. It must match the CONNECTION parameter specified in the SESSIONS definition. See Figure 4-3 on page 70.

- **NETNAME**

  If this is specified, then the connection is configured to use a *specific pipe*. If it is left blank, then the EXCI connection will use a *generic pipe*. If using specific pipes, then this must correspond to the name of the pipe as also specified in the EXCI client program. This is specified on the `user_name` parameter of an Initialize_User EXCI call. When using the CICS TG for z/OS, this value is passed to the EXCI using the DFHJVPIPE environment variable, the setting of which is described in Figure 7-5 on page 149.

  If the NETNAME parameter is not specified, then a *generic pipe* will be used, which is essentially an unnamed pipe. Generally it is best to use a specific pipe, since this aids in management of multiple connections and in problem determination.

Note that when a single CICS TG for z/OS region is configured to connect to more than one CICS region, you must use the same NETNAME for all of the EXCI connections.

> **Note:** It is quite possible to use the same DFHJVPIPE for multiple address spaces (that is, multiple Gateway daemons). However, in this way if all the address spaces connect to the same region, they will share the same connection, and this may adversely affect your ability to monitor traffic across the connection, and will also require a higher RECEIVECOUNT to be defined in the SESSIONS definition.

- **ACCESSMETHOD**

  For EXCI connections, this must be specified as inter-region communication (IRC). EXCI is implemented through the IRC program (DFHIRP), using the supervisor call (SVC) mode of DFHIRP (as opposed to cross memory). DFHIRP can use the cross-system coupling facility (XCF) MRO links, but cannot use MVS cross-memory services (XM).

- **PROTOCOL**

  For EXCI, this must be specified as EXCI.

- **CONNTYPE**

  This is the type of EXCI connection used for non-CICS jobs to communicate with a CICS region on z/OS. It is a type of MRO request and must be specified as either specific or generic.

  – SPECIFIC**:** An MRO link with one or more sessions dedicated to a single user in a client program. For a specific connection, the NETNAME attribute is mandatory.

  – GENERIC**:** An MRO link with a number of sessions to be shared by multiple EXCI users. Only one generic EXCI connection can be installed in each region. For a generic connection, the NETNAME attribute must be left blank.

- **ATTACHSEC**

  Controls the security checks made using the flowed user ID.

  – IDENTIFY**:** Specifies to CICS that a user ID will be flowed on every request, but no password is expected as CICS will *trust* the user ID as having been already authenticated. When using the CICS TG for z/OS the user ID will be either the user named in the `ECIRequest` object or, if null, the user ID of the thread the ECI request runs under.

When using the CICS TG for z/OS (using either the Gateway daemon or WebSphere Application Server), for security reasons this user ID and password should be verified with RACF before the EXCI request is made.

– LOCAL: Specifies that a user ID will not be flowed by the remote system, and instead only the link user ID (if specified) will be used. If no link user ID is supplied, then all requests will be run under the CICS default user ID as specified in the DFLTUSER SIT parameter.

For more details on CICS security, refer to Chapter 6, “CICS TG security scenarios” on page 99.

4.2.2 EXCI SESSIONS definitions

For each EXCI connection definition, it is necessary to use a SESSIONS definition to define the parameters of the SESSIONS being used. Figure 4-3 shows the SESSIONS definition used for our connection CTG5 to our CICS TG region SCSCTG5.

```
 OVERTYPE TO MODIFY                                        CICS RELEASE = 0530
   CEDA  ALter Sessions( EXCGSESS )
    Sessions       : EXCGSESS
    Group          : CTG5
    DEscription  ==> CICS TG SESSIONS DEFINITION
   SESSION IDENTIFIERS
    Connection   ==> CTG5
    SESSName     ==>
    NETnameq     ==>
    MOdename     ==>
   SESSION PROPERTIES
    Protocol     ==> Exci                Appc | Lu61 | Exci
    MAximum      ==> 000 , 000           0-999
    RECEIVEPfx   ==> C5
    RECEIVECount ==> 100                 1-999
    SENDPfx      ==>
    SENDCount    ==>                     1-999
    SENDSize     ==> 04096               1-30720
    RECEIVESize  ==> 04096               1-30720
PRESET SECURITY
    USERId       ==>
                                                        SYSID=PAA6 APPLID=SCSCPAA6
```

Figure 4-3 EXCI sessions definition

The key parameters in the SESSIONS definition are:

- **CONNECTION**

  Associates this SESSIONS definition with the CONNECTION name CTG5.

- **PROTOCOL**

  Indicates the type of sessions to be used for an intercommunication connection. Sessions used by EXCI must specify EXCI for the protocol.

- **RECEIVEPFX**

  Specifies a one- or two-character prefix that CICS is to use as the first one or two characters of the receive session names. CICS creates the remainder of the four-character names from the alphanumeric characters A through Z, and 1 through 9. These two- or three-character identifiers begin with the letters AAA, and continue in ascending sequence until the number of session entries reaches the limit set by the RECEIVECOUNT value (in our example C5AA, C5AB, C5AC, and so on). The default receive prefix is (<).

- **RECEIVECOUNT**

  Specifies the number of sessions that receive before sending. Since all EXCI sessions are receive sessions, this defines the number of pipes that can simultaneously be used. The EXCI itself imposes a maximum limit of 100 pipes per EXCI address space, therefore the RECEIVECOUNT should not exceed 100 unless multiple EXCI clients are connected to the same CICS region using the same EXCI connection. This could be the case, for instance, if you were using cloned CICS TG regions connecting to the same CICS region.

- **SENDCOUNT**

  Should be left blank for EXCI sessions. Since EXCI sessions can only receive, there are no send sessions.

- **USERID**

  This is the preset user identifier to be used for link security checking. This provides an additional security check for each transaction in addition to the flowed user ID. If you do not specify a preset user ID for link security, CICS uses the user ID of the remote system. For further details on link security, refer to Chapter 6, “CICS TG security scenarios” on page 99

4.2.3 EXCI options table: DFHXCOPT

The EXCI options table, DFHXCOPT, enables you to specify a number of parameters that the EXCI requires. There is no suffixed version of this program, so the first DFHXCOPT table located in the STEPLIB concatenation or linklist will be loaded. Example 4-1 shows the default parameters for the DFHXCOPT macro. Assembly of DFHXCOPT is accomplished by the use of the DFHAUPLE proc in CICSTS22.CICS.SDFHPROC.

Example 4-1 EXCI options table, DFHXCOPT

```
DFHXCO TYPE=CSECT,
               TIMEOUT=0,              No timeout
               TRACE=OFF,              Trace entries
               TRACESZE=1024,          Trace table
               DURETRY=30,             Retry SDUMPS for 30 seconds
               TRAP=OFF,               DFHXCTRA - OFF
               GTF=ON,                 GTF - ON
               MSGCASE=MIXED,          Mixed case messages
               CICSSVC=0,              EXCI will obtain CICS SVC number
               CONFDATA=SHOW,          Show user commarea data in trace
               SURROGCHK=YES           Perform surrogate-user check
         END   DFHXCOPT
```

The meanings of the significant parameters are:

- **TRACE**

  Set to OFF. This specifies that EXCI tracing is not required, but exception trace entries are always written to the internal trace table.

- **CICSSVC**

  Specifies the CICS type 3 SVC number to be used for MRO communication. EXCI must use the same SVC number that is in use by the CICS MRO regions that reside in the MVS image where the client program is running. The default of zero indicates that EXCI will obtain the CICS SVC number from MVS by means of an MVS VERIFY command. Using this default is highly recommended wherever possible. If you allow this parameter to default, and EXCI requests the SVC from MVS, then the request will fail if no CICS region has logged on to DFHIRP.

- **CONFDATA**

  SHOW will allow user data to be traced and not suppressed. If you want trace entries that are normally written to the EXCI trace table to also be written to the generalized trace facility (GTF), then specify GTF=ON.

- **SURROGCHK**

  Defaults to YES, meaning that a surrogate security check will be performed in the EXCI client address space if the flowed user ID is different from the user ID of the client address space. This occurs regardless of the CICS security settings. This means that if you flow a user ID in an EXCI request (including flowing a user ID with an External Call Interface [ECI] request using the CICS TG for z/OS), you will need to either add a RACF surrogate profile, or disable surrogate security checks by setting this parameter to NO. For details on how to add surrogate profiles, refer to “SURROGAT security profiles” on page 109.

Refer to the *CICS External Interfaces Guide,* SC33-1944, for a detailed description of the DFHXCOPT parameters.

4.2.4 EXCI user-replaceable module: DFHXCURM

The user-replaceable module DFHXCURM is invoked on every EXCI Allocate_Pipe request, and also after detection of any EXCI retryable errors. This will occur under one of three circumstances:

- The target CICS region is not available.
- There are no pipes available on the target CICS region.
- IRC has not been active since the last initial program load (IPL).

DFHXCURM can be used to remove the affinity of an EXCI request to a given CICS region by dynamically modifying the APPLID in the EXCI request. It can also be used to provide limited workload balancing of EXCI requests across multiple CICS regions from the CICS TG for z/OS. For further details on this subject, refer to *Workload Management for Web Access to CICS*, SG24-6118.

4.2.5 Transactional EXCI

Transactional EXCI works together with MVS resource recovery services (RRS), and allows multiple EXCI calls to become part of one logical unit of work. This now means that the CICS TG for z/OS can be used by other transactional systems (such as WebSphere V4.0 for z/OS) to make transaction requests to CICS, which can be coordinated using a two-phase commit mechanism between the two systems.

> **Restriction:** EXCI requests can only be transactional if the address space using the EXCI and the CICS region execute in the same z/OS image. This is a restriction imposed by the RRS support in CICS.

To implement transactional EXCI, you must specify the following parameters:

- RRMS=YES in CICS SIT parameter

- CTG_RRMNAME as a CICS TG environment variable

  This is the name that the CICS TG registers with RRS for transactional EXCI requests to CICS. You will only need to modify this if using transactional EXCI requests. For further details on how we implemented transactional EXCI with our CICS TG for z/OS, refer to Chapter 7, "TCP connections to the Gateway daemon on z/OS" on page 133.

4.3 EXCI definitions for the CICS TG

The following environment variables are specific to the CICS TG for z/OS. We specified these parameters in the PDS member named on our STDENV card in the CICS TG started task, but they can also be set in the ctgenvvar HFS file, among other places. For further details on how the settings of these variables are controlled, refer to Figure 7-5 on page 149. For step-by-step details on how we configured our CICS TG for z/OS, refer to Chapter 7, "TCP connections to the Gateway daemon on z/OS" on page 133.

The parameters we used in our CICS TG for z/OS are as follows:

- DFHJVPIPE

  This parameter names the EXCI pipe in use by the CICS TG (when using a specific EXCI connection). If not specified, the CICS TG will use a generic pipe. We set this parameter to the name of our CICS TG region (SCSCTG5), to help with management and problem diagnosis.

Note: The DFHJVPIPE value must match the NETNAME parameter in the CICS CONNECTION definition.

- EXCI_LOADLIB

  This is the name of the library containing the EXCI load modules (without its high-level qualifier). This will usually be SDFHEXCI, and so does not usually need modifying.

- EXCI_OPTIONS

  This is a user library that contains the EXCI options to be used if you have a tailored version of the EXCI options table (DFHXCOPT). For more details, refer to "SURROGAT security checking and DFHXCOPT" on page 125.

4.4 Problem determination

In this section, we discuss the different tools available for performing problem resolution for CICS EXCI connections.

4.4.1 Tips and utilities

In configuring our EXCI connections in this project, we used the following utilities to aid us in problem determination.

EXCI sample batch program

When configuring our EXCI connections, we decided it was easiest to test our connections using the CICS EXCI sample batch program DFH0CXCC. This way we could be sure our EXCI connections were working, before we implemented our CICS TG for z/OS.

DFH0CXCC is a simple batch program, written in COBOL that uses the EXCI to call the CICS assembler program, DFH$AXCS. Further information on implementing this application can be found in the “Sample Applications Appendix” in the manual *CICS Resource Definitions,* SC33-1684.

> **Tip:** DFH0CXCC calls the CICS program DFH$AXCS using a generic pipe (since the `APPLICATION` variable is set to zero). To properly test connectivity when using specific pipes with the CICS TG for z/OS, we suggest you modify the source for DFH0CXCC to use the pipe named in your CICS TG DFHJVPIPE variable and test using this specific pipe. In our example, we changed it to SCSCTG5.

CEDF

We found it useful to use the CICS execution diagnostic facility in our tests. It is possible to turn CEDF on in one of two ways when using EXCI requests:

CEDF CTG5 This causes all incoming transaction attaches on the *connection* named CTG5 to be suspended and debugged by the CICS execution diagnostic facility program, DFHEDFP, on the terminal in use.

CEDX CSMI This causes all transaction attaches for the CSMI *transaction* to be suspended and debugged by the CICS execution diagnostic facility program, DFHEDFP, on the terminal in use.

Note that use of the CEDF or CEDX causes all transactions using the named connection or transaction to be suspended, so use with care and remember to switch it off when you are finished debugging. To turn off CEDX for the CSMI transaction, use the syntax `CEDX CSMI,OFF`.

Connection status

When diagnosing errors in EXCI, it is best to check first that the CICS resources are open and available. The following transactions can be used to verify this.

Verify IRC

The first check should usually be to view the status of IRC, which must be open for EXCI to function. Issue the command `CEMT INQ IRC`. The output is shown in Figure 4-4.

```
 INQ IRC
 STATUS:  RESULTS - OVERTYPE TO MODIFY
  Irc Ope









                                                SYSID=PJA1 APPLID=SCSCPJA1
  RESPONSE: NORMAL                             TIME:  19.28.44  DATE: 05.28.02
PF 1 HELP       3 END       5 VAR         7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Figure 4-4 Displaying the IRC status in CICS

View the connection definition

To view the status of all connections, issue the command `CEMT INQ CONN`. The output is shown in Figure 4-5.

```
 INQ CONN
 STATUS:  RESULTS - OVERTYPE TO MODIFY
 Con(CTG5) Net(SCSCTG5 )     Ins     Irc Exci









                                              SYSID=PJA1 APPLID=SCSCPJA1
  RESPONSE: NORMAL                       TIME:  19.22.54  DATE: 05.28.02
PF 1 HELP       3 END       5 VAR        7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Figure 4-5 Displaying connections in CICS

The inquiry of connection definitions is useful because it displays all the defined connections and also the netnames for each connection. The status of all EXCI connections should be `Ins`, although this does not mean any sessions are acquired.

View status of EXCI pipes

To display the individual status of each session, we entered the command:

```
CEMT INQ NETNAME(SCSCTG5)
```

where `SCSCTG5` is the netname for our connection (as defined in our DFHJVPIPE variable). The output is shown in Figure 4-6 on page 78.

```
  INQ NETNAME(SCSCTG5)
 STATUS:  RESULTS - OVERTYPE TO MODIFY
  Net(SCSCTG5 ) Tra(CSMI) Pri( 000 ) Aut Ins Ati Tti Ses
     Ter(C51 )       Tas(0001608) Rem(CTG5)
  Net(SCSCTG5 )           Pri( 000 ) Aut Ins Ati Tti Ses
     Ter(C52 )                      Rem(CTG5)
  Net(SCSCTG5 )           Pri( 000 ) Aut Ins Ati Tti Ses
     Ter(C53 )                      Rem(CTG5)
  Net(SCSCTG5 )           Pri( 000 ) Aut Ins Ati Tti Ses
     Ter(C54 )                      Rem(CTG5)
  Net(SCSCTG5 )           Pri( 000 ) Aut Ins Ati Tti Ses
     Ter(C55 )                      Rem(CTG5)
  Net(SCSCTG5 )           Pri( 000 ) Aut Ins Ati Tti Ses
     Ter(C56 )                      Rem(CTG5)
  Net(SCSCTG5 )           Pri( 000 ) Aut Ins Ati Tti Ses
     Ter(C57 )                      Rem(CTG5)
  Net(SCSCTG5 )           Pri( 000 ) Aut Ins Ati Tti Ses
     Ter(C58 )                      Rem(CTG5)
+ Net(SCSCTG5 )           Pri( 000 ) Aut Ins Ati Tti Ses
     Ter(C59 )                      Rem(CTG5)

                                              SYSID=PJA2 APPLID=SCSCPJA1
  RESPONSE: NORMAL                       TIME:  00.59.01  DATE: 06.17.02
PF 1 HELP       3 END       5 VAR        7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Figure 4-6 Displaying netnames

The status of `Ins` for each session indicates the status of each pipe. The transaction that CSMI displayed for the first session, C51, indicates that an EXCI request is currently active on that given session (we achieved this result by suspending the CSMI mirror task by using CEDF). The number of sessions in the display is a useful indicator of the number of sessions defined in the EXCI connection. In our example we had defined a RECEIVECOUNT of 10 in our SESSIONS definition, and a RECEIVEPFX of C5, so `CEMT INQ NETNAME` displayed 10 sessions named C51 to C510.

View status of EXCI calls

To view the status of active EXCI calls, issue the command `CEMT INQ EXCI`. The results are shown in Figure 4-7.

```
 INQ EXCI
 STATUS:  RESULTS
  Exc(SCSCTG55.*OMVSEX. - SC66          ) Tas(0001608)






                                             SYSID=PJA1 APPLID=SCSCPJA1
  RESPONSE: NORMAL                       TIME:  19.30.26  DATE: 05.28.02
PF 1 HELP       3 END       5 VAR        7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Figure 4-7 Displaying an EXCI connection

The results in Figure 4-7 show that there is currently an EXCI request active (task 1608) associated with the region SCSCTG55, which is one of the processes used by our CICS TG, SCSCTG5.

XCF group membership

An important consideration when the CICS TG is used with the EXCI is the usage of slots in the DFHIR000 XCF (cross-system coupling facility) group in the sysplex couple data set. This XCF group is used by CICS when two address spaces on different LPARs communicate by means of MRO or EXCI. The maximum possible limit for group membership is 1023 members per group (it was raised from 511 by APAR OW21511), but the actual limit is defined in the MAXMEMBER parameter when the couple data set is formatted. To display this limit, the MVS command `/D XCF,COUPLE` can be used. The output of this command when we moved our Gateway region SCSCTG5 to another LPAR in our sysplex is shown in Example 4-2. You can see that all the CICS regions in our sysplex are listed as individual members (such as SCSCPJA1), but our Gateway region has a member named for the thread under which the EXCI is running (T02CEF40SC66), and not for the Gateway address space.

Example 4-2 Displaying XCF connections

```
IXC332I  13.47.38  DISPLAY XCF 105
 GROUP DFHIR000:  SCSCCOB1         SCSCERW1         SCSCERW2
                  SCSCERW3         SCSCLSA5         SCSCPAAS
                  SCSCPAAY         SCSCPAA2         SCSCPAA6
                  SCSCPAGV         SCSCPAME         SCSCPJA1
                  SCSCPJA2         SCSCPJA4         T02CEF40SC66
```

4.4.2 Tracing

EXCI outputs trace to an internal trace table and an external MVS GTF data set. The internal trace table resides in the non-CICS MVS batch region. EXCI maintains a separate trace table for each user task control block (TCB) in an EXCI application program. Trace data is formatted and included in any dumps EXCI produced. The trace entries are listed in the manual *CICS Trace Entries,* SC34-5446.

EXCI produces MVS SYSM dumps for some error conditions and MVS SDUMPs for more serious conditions. These dumps contain all the external CICS interface control blocks as well as trace entries. You can use the MVS interactive problem control system (IPCS) to format these dumps. For details on using IPCS, refer to *CICS Operations and Utilities Guide,* SC33-1685.

To use GTF for EXCI tracing, GTF user tracing must be active, GTF must be started in the MVS image, and you must specify GTF=ON in the DFHXCOPT options table. If you use GTF trace for both the CICS server region and the EXCI region, the trace entries are interleaved, which can help you with problem determination in the CICS-EXCI environment. The external CICS interface does not support any form of auxiliary trace. Actual examples of how to use EXCI tracing are provided in 7.4.2, "Tracing" on page 170.

Gateway trace

When using the CICS TG for z/OS, there are essentially four different types of tracing you can use: EXCI tracing, normal Gateway trace, extended Gateway trace, and Gateway JNI trace. We found the most useful traces were the normal Gateway trace, which provides a quick snapshot view of an ECI request, and JNI trace, which provides return codes for all the EXCI calls. Actual examples of how we used each of these are explained in detail in 7.4.2, "Tracing" on page 170.

5

TCP/IP connections to CICS

In this chapter, we describe how we configured a TCP/IP connection from the CICS Transaction Gateway (CICS TG) on Windows 2000 to our CICS Transaction Server (CICS TS) V2.2 region. We show only Windows 2000, but these steps can be used to configure all the platforms except z/OS, since there are no major differences. Our configuration is illustrated in Figure 5-2 on page 83.

5.1 Introduction to ECI over TCP/IP

ECI over TCP/IP is a communications mechanism that provides the ability to connect from the CICS TG to a CICS TS region over an IP network. It utilizes the support for ECI over TCP/IP in CICS TS V2.2 to send CICS Universal Client or CICS Transaction Gateway ECI requests over an IP network.

Figure 5-1 shows the various types of ECI connections currently supported by CICS. A SNA connection can be made via VTAM into CICS. Compared to SNA, TCP/IP configuration is more straightforward and can be flowed over an IP network. In order to flow APPC over an IP network, a client uses TCP62 to communicate with AnyNet, which performs protocol conversion and then communicates with CICS via VTAM. TCP/IP does not incur the protocol conversion overhead of TCP62 and again is more straightforward to configure, as neither AnyNet nor VTAM is required.

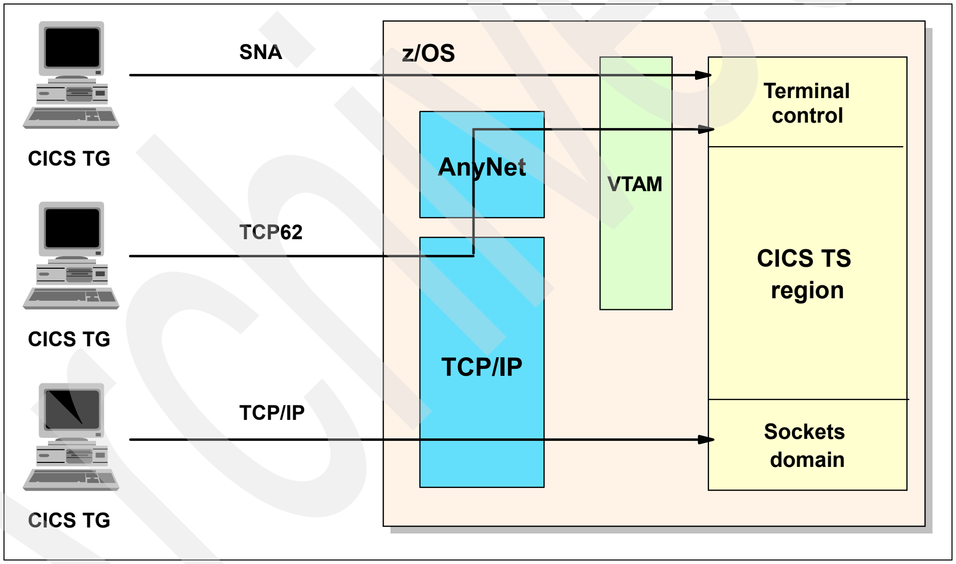


Figure 5-1 Overview of ECI connection types

In Figure 5-2 on page 83 we illustrate an overview of the TCP/IP environment that we configure in this chapter.

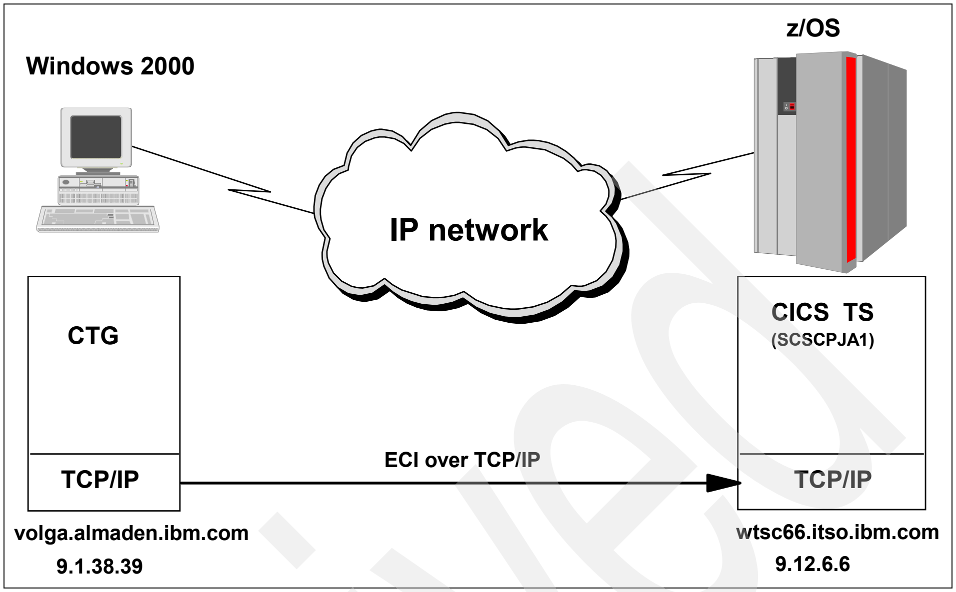


Figure 5-2 CICS TG TCP/IP configuration

5.1.1 Software checklist

We used the levels of software shown in Table 5-1.

Table 5-1 Software levels, EXCI

Client workstation	z/OS
Windows 2000 Service Pack 2	z/OS V1.2
CICS Transaction Gateway for Windows V5.0	CICS Transaction Server V2.2
IBM Java Development Kit V1.3.0	

5.1.2 Definitions checklist

The definitions we used to configure this scenario are summarized in Table 5-2. Before you configure the products, we recommend that you decide on definitions for each parameter listed. Reference keys shown in the Key column are assigned to definitions that should contain the same value in more than one product. The Example column shows the values we used in our configuration.

Table 5-2 Definitions checklist for ECI over TCP/IP

Key	Communications Server for z/OS	CICS region	CICS TG server definition	Example
1	hostname		hostname	wtsc66.itso.ibm.com
2		PORT	port	8018
		APPLID		SCSCPJA1
		TCPIPSERVICE		PJA1TCP
		GROUP		PJA1GRP

In addition, we also had to deploy our CICS COBOL application EC01 and configure a DFHCNV data conversion template in CICS for use by this program. Refer to Appendix A, "DFHCNV and CICS data conversion" on page 329 for more details.

5.2 TCP/IP definitions for CICS

To configure a CICS TCP/IP listener to handle ECI requests, we first had to confirm the following definitions were installed into CICS:

- SIT parameter: TCPIP=YES
- CICS-supplied Transient Date queue CIEO, in group DFHDCTG
- Transaction CIEP in group DFHIPECI
- Program DFHIEP in group DFHIPECI

All of these definitions are included in the list DFHLIST and so were activated in our CICS region. The SIT parameter ISC=YES is not required for operation of ECI over TCP/IP.

z/OS TCP/IP definitions

The TCP/IP address and host name (**1**) for the z/OS system are defined by default in the PROFILE.TCPIP and TCPIP.DATA data sets. In our configuration the data set was TCPIPMVS.SC66.TCPPARMS and it has members TCPPROF and TCPDATA. These members have a lot of information, but we are only interested in the HOSTNAME, and DOMAINNAMEORIGIN parameters in the TCPDATA member and the HOME parameter in the TCPPROF member. In our configuration, these parameters were as follows:

HOSTNAME wtsc66 is the host name of our z/OS system

DOMAINNAMEORIGIN itso.ibm.com is the domain name

HOME 9.12.6.6 is the TCP/IP address

> **Tip:** You can also use the TSO command **HOMETEST** to verify the IP configuration on your z/OS system.

CICS TCP/IP listener

In order to add a TCP/IP listener to CICS, we defined a TCPIPSERVICE called PJA1TCP in the group PJA1GRP. PJA1GRP was in the startup GRPLIST for our CICS region, so the listener will start when CICS is started. We used the following command from a CICS terminal to define the listener (see Figure 5-3).

```
CEDA DEF TCPIPSERVICE(PJA1TCP) GROUP(PJA1GRP)
```

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0620
 CEDA  DEFine TCpipservice( PJA1TCP )
  TCpipservice   : PJA1TCP
  GROup          : PJA1GRP
  Urm          ==>
  POrtnumber   ==> 08018           1-65535
  STatus       ==> Open            Open | Closed
  PRotocol     ==> Eci             Iiop | Http | Eci
  TRansaction  ==> CIEP
  Backlog      ==> 00100           0-32767
  Ipaddress    ==> ANY
  SOcketclose  ==> No              No | 0-240000 (HHMMSS)
 SECURITY
  SSl          ==> No              Yes | No | Clientauth
  Certificate  ==>
  AUthenticate ==>                 No | Basic | Certificate | AUTORegister
  ATtachsec    ==> Local           Local | Verify
                                                     SYSID=PJA1 APPLID=SCSCPJA1
   DEFINE SUCCESSFUL                              TIME:  15.30.02  DATE: 02.183
```

Figure 5-3 ECI over TCP/IP TCPIPSERVICE definition

The fields relevant to ECI on the TCPIPSERVICE definition are as follows:

PORTNUMBER 2 The port on which the TCP/IP service listens.

PROTOCOL The protocol of the service. This should be ECI.

TRANSACTION The transaction that is run on CICS to handle the incoming ECI requests. This is CIEP for ECI.

BACKLOG The number of TCP/IP connections that are queued in TCP/IP before TCP/IP starts to reject incoming requests. We set this to 100.

IPADDRESS The dotted decimal IP address on which the TCPIPSERVICE listens. Since we had two IP stacks, we specified ANY so that the CICS TCP/IP listener would listen on both addresses.

SOCKETCLOSE Specifies if, and for how long, CICS should wait before closing the socket, after issuing a receive for incoming data on that socket. For ECI connections, we recommend NO. This will ensure that the connection from the Client daemon always remains open.

ATTACHSEC Specifies the level of attach-time security required for TCP/IP connections to CICS Clients. For testing purposes, we specified LOCAL, which means that CICS does not require a user ID or password from clients.

We installed our TCPIPSERVICE definition using the command:

```
CEDA INS TCPIPSERVICE(PJA1TCP) GROUP(PJA1GRP).
```

We used the command `CEMT I TCPIPS` to display the active service (Figure 5-4).

```
I TCPIPS
STATUS:  RESULTS - OVERTYPE TO MODIFY
  Tcpips(PJA1TCP ) Bac( 00128 ) Con(0003) Por(08018) Eci Nos
        Ope Tra(CIEP)                Ipa(9.12.6.29      ) Wai







                                                   SYSID=PJA1 APPLID=SCSCPJA1
 RESPONSE: NORMAL                             TIME:  15.44.37  DATE: 07.02.02
PF 1 HELP       3 END       5 VAR       7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Figure 5-4 TCP/IP listener status

5.3 TCP/IP definitions for the CICS TG

We used the Configuration Tool to define the settings for TCP/IP communication. The Configuration Tool generates the CTG.INI file, which is located in the \bin subdirectory. This file is read by the Client daemon when started and used to establish a connection to a CICS Server. We defined the parameters as shown in Figure 5-5.

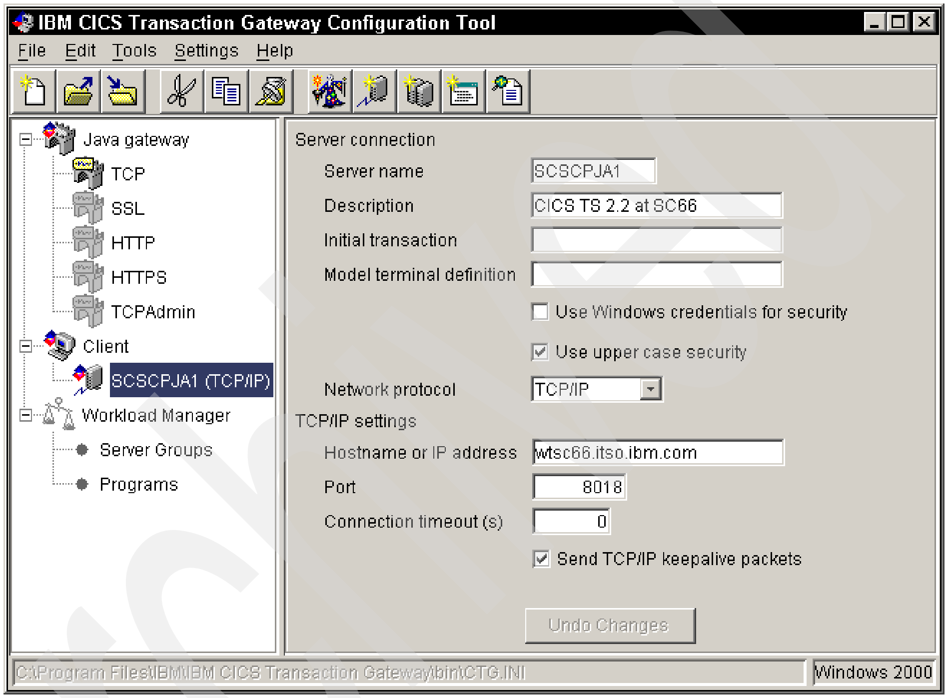


Figure 5-5 TCP/IP server definition in the Configuration Tool

Server name An arbitrary name for your CICS server connection. We specified the region name SCSCPJA1.

Description An arbitrary description for the CICS server.

Network protocol The protocol for communication with the CICS server, in our example, TCP/IP.

Hostname 1 The host name of the z/OS system on which our CICS server is listening, in our case wtsc66.itso.ibm.com.

Port 2 The port on which our ECI over TCP/IP service is listening, in our case port 8018.

This creates the lines shown in Example 5-1 on page 88 in the CTG.INI file.

Example 5-1 Client daemon server definition in CTG.INI

```
SECTION SERVER = SCSCPJA1
    DESCRIPTION=CICS TS 2.2 at SC66
    UPPERCASESECURITY=Y
    USENPI=N
    PROTOCOL=TCPIP
    NETNAME=wtsc66.itso.ibm.com
    PORT=8018
    CONNECTTIMEOUT=0
    TCPKEEPALIVE=Y
ENDSECTION
```

Testing the configuration

To test our configuration we used the CICS TG sample VBScript application ecib1.vbs. This is installed in the CICS TG /samples/vb/script subdirectory.

Because the sample is a script, no compilation is necessary. The script can be run directly from any Windows 2000 workstation using the built-in Windows Script Host support. To execute, just double-click the script file on the Windows desktop, or type the name of the script file at the command prompt. The application flows an ECI request to a connected CICS region through a specified server connection using the Client daemon and invokes the transaction EC01 (Figure 5-6). The CICS region is selected from a window.

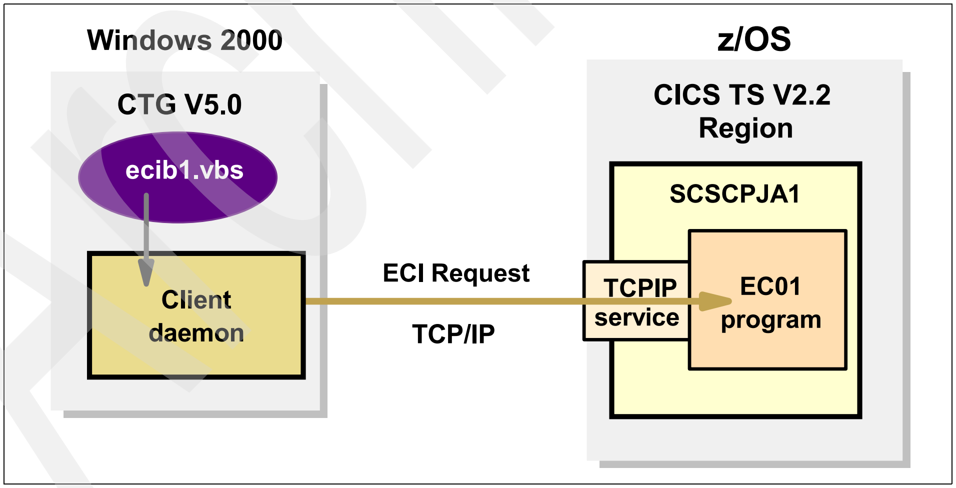


Figure 5-6 TCP/IP connection to CICS testing overview

After we installed and configured the software components illustrated in Figure 5-6, we tested our configuration as follows:

1. We checked basic IP connectivity from our Windows workstation to the z/OS system using the **ping** command from a Windows 2000 prompt. As shown in Example 5-2, we successfully received a reply from z/OS.

Example 5-2 Output from the ping command on Windows

```
C:\>ping wtsc66.itso.ibm.com

Pinging wtsc66.itso.ibm.com [9.12.6.6] with 32 bytes of data:

Reply from 9.12.6.6: bytes=32 time=130ms TTL=52
Reply from 9.12.6.6: bytes=32 time=120ms TTL=52
Reply from 9.12.6.6: bytes=32 time=120ms TTL=52
Reply from 9.12.6.6: bytes=32 time=120ms TTL=52

Ping statistics for 9.12.6.6:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 120ms, Maximum =  130ms, Average =  122ms
```

2. We started the CICS region SCSCPJA1 on z/OS.
3. Using the ScanPort utility, we then checked that CICS was listening on the TCP/IP port configured in the CICS TG server connection definition. See page 52 in Chapter 3 for more details on how we used the ScanPort utility.
4. We started the CICS TG connection to the CICS region using the command `CICSCLI /S=SCSCPJA1`. (`SCSCPJA1` is the Server name given in the Client configuration step in Figure 5-5 on page 87.)
5. We checked the status of the connection to the CICS region using the command `CICSCLI /L`. The connection status is available, as shown in Example 5-3.

Example 5-3 Checking Client daemon connection status

```
CCL8001I CICSCLI - CICS Client Control Program
CCL0002I (C) Copyright IBM Corporation 1994, 2002.  All rights reserved.
CCL8041I The CICS Client is using the following servers:
CCL8042I Server 'SCSCPJA1' (using 'TCPIP' to 'wtsc66.itso.ibm.com') is
available
```

6. We launched the ecib1.vbs test application. This showed the window in Figure 5-7 on page 90, which displays the server connections defined to the Client daemon. We entered `1` to select SCSCPJA1 and clicked **OK**.

Figure 5-7 ecib1.vbs server selection window

7. We clicked **OK** on the window that appeared asking for a user name and password, since SCSCPJA1 did not require a user ID.

The application then sent an ECI request to our CICS region and displayed the data returned in the COMMAREA from EC01 in the window shown in Figure 5-8.

Figure 5-8 ecib1.vbs results

5.4 Problem determination

In this section, we document information we learned while configuring this scenario, as well as further information on problem determination and tracing.

5.4.1 Tips and utilities

In this section, you will find useful commands and utilities for debugging any problems with your configuration. You will also find some additional information about topics discussed in this chapter.

Client daemon TCP/IP

For problem determination at the client side, it is best to check the CICSCLI.LOG file under the CICS TG bin directory for error messages. If this does not give enough information, you should start the connection with trace.

CICS TS

For problem determination with TCP/IP connections into CICS TS, you should refer to the following logs:

- z/OS system log (SDSF.LOG) for TCP/IP information messages
- CICS console and CSMT logs for CICS messages

Windows TCP/IP status

To check the status of TCP/IP as used by the Client daemon on the workstation, you can issue the **netstat /a /n** command to view socket usage as shown in Example 5-4.

Example 5-4 Client side netstat information

```
C:\>netstat /a /n
Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    9.1.38.39:1142         9.12.6.29:23           ESTABLISHED
  TCP    9.1.38.39:1202         9.12.6.6:8018          ESTABLISHED
  TCP    9.1.38.39:1148         9.12.6.6:23            ESTABLISHED
  TCP    9.1.38.39:1323         9.17.186.68:1533       ESTABLISHED
```

This shows that the Client daemon is using the TCP port 1202 on our workstation (9.1.38.39), and that a session is established using a TCP connection with the z/OS system (9.12.6.6).

z/OS TCP/IP status

The D TCPIP command issued from SDSF displays the status of TCP/IP on the z/OS system as shown in Example 5-5.

Example 5-5 Status of TCP/IP

```
D TCPIP
RESPONSE=SC66
 EZAOP50I TCPIP STATUS REPORT 513
 COUNT   TCPIP NAME   VERSION    STATUS
 -----   ----------   --------   ------------------------
     1   TCPIPOE      CS V1R2    ACTIVE
     2   TCPIPMVS     CS V1R2    ACTIVE
 *** END TCPIP STATUS REPORT ***
 EZAOP41I 'DISPLAY TCPIP' COMMAND COMPLETED SUCCESSFULLY
```

In our example, we were using the TCP/IP stack TCPIPMVS on our z/OS system. However, because we specified ALL for the IPADDRESS of our TCPIPSERVICE, our CICS region is listening on both TCP/IP stacks.

To display the TCP/IP sockets in use by CICS, you can use the TSO NETSTAT command as shown in Example 5-6. This will give you similar information that you got from the client side. The output shows our CICS region listening on port 8018.

Example 5-6 MVS NETSTAT

```
EZZ2350I MVS TCP/IP NETSTAT CS V1R2    TCPIP NAME: TCPIPMVS         18:25:59
EZZ2585I User Id  Conn     Local Socket       Foreign Socket         State
EZZ2586I -------  ----     ------------       --------------         -----
EZZ2587I SCSCPJA1 000085F2 0.0.0.0..8011      0.0.0.0..0             Listen
EZZ2587I SCSCPJA1 000085F4 0.0.0.0..8018      0.0.0.0..0             Listen
EZZ2587I SCSCPJA1 000085F0 0.0.0.0..8010      0.0.0.0..0             Listen
```

To display the TCP/IP sockets in use by CICS on the TCP/IP stack TCPIPOE, you can use the OMVS command **netstat /a** as shown in Example 5-7. The output shows our CICS region listening on port 8018 on the TCPIPOE stack.

Example 5-7 OMVS netstat

```
MVS TCP/IP onetstat CS V1R2       TCPIP Name: TCPIPOE           14:48:01
User Id  Conn     Local Socket           Foreign Socket          State
-------  ----     ------------           --------------          -----
PMAPOE1  00000037 0.0.0.0..111           0.0.0.0..0              Listen
SCSCPJA1 00000B09 0.0.0.0..8078          0.0.0.0..0              Listen
SCSCPJA1 00000D90 0.0.0.0..8018          0.0.0.0..0              Listen
SCSCPJA1 00000B03 0.0.0.0..8010          0.0.0.0..0              Listen
```

Traceroute

If the **ping** command fails to reach a host, then the **tracert** command can be used to discover more about the problem. Like **ping**, **tracert** tests connectivity between two network hosts; however, **tracert** lists each router in the network between the machine running the command and the destination host. Sometimes an inability to contact a host is due to one of these routers not forwarding IP traffic, so **tracert** can be used to discover which router is preventing a connection to a remote host.

In Example 5-8 on page 93 we traced the route from our Windows workstation to our z/OS system using the command **tracert wtsc66.itso.ibm.com** from a Windows command prompt.

Example 5-8 tracert command

```
C:\>tracert wtsc66.itso.ibm.com

Tracing route to wtsc66.itso.ibm.com [9.12.6.6]
over a maximum of 30 hops:

 1   <10 ms   <10 ms   <10 ms  router-38.almaden.ibm.com [9.1.38.1]
 2   <10 ms   <10 ms   <10 ms  Almbda-ldb.almaden.ibm.com [9.1.80.139]
 3   <10 ms   <10 ms   <10 ms  Sjalmsca.almaden.ibm.com [9.1.80.6]
 4   <10 ms   <10 ms   <10 ms  9.43.4.14
 5    60 ms    60 ms    60 ms  GA008-R04-6509.wan.ibm.com [9.64.23.165]
 6    60 ms    60 ms    70 ms  GA008-R02-12008.wan.ibm.com [9.64.18.13]
 7   120 ms   120 ms   120 ms  NY017-R02-12008.wan.ibm.com [9.64.1.33]
 8   120 ms   120 ms   110 ms  POK-W-6509-R-0034-010-1-VL556.pok.ibm.com [9.56.
4.3]
 9   120 ms   130 ms   120 ms  POK-W-4908-R-0001-010-1-VL256.pok.ibm.com [9.56.
1.3]
10   120 ms   120 ms   120 ms  9.56.2.9
11   120 ms   111 ms   120 ms  9.56.2.46
12   120 ms   120 ms   130 ms  pokp6509r008a.pok.ibm.com [9.56.126.6]
13   121 ms   120 ms   110 ms  wtsc66.itso.ibm.com [9.12.6.6]

Trace complete.
```

As shown, the connection to our z/OS system is working and it uses 12 routers in the TCP/IP network. To demonstrate what might happen when a host is unaccountable, we traced the route to an Internet host that was down, as shown in Example 5-9.

Example 5-9 tracert to an unavailable host

```
C:\>tracert -h 3 216.105.167.65

Tracing route to 216.105.167.65 over a maximum of 3 hops

 1   <10 ms   <10 ms   <10 ms  router-38.almaden.ibm.com [9.1.38.1]
 2     *        *        *     Request timed out.
 3     *        *        *     Request timed out.

Trace complete.
```

As shown, router-38.almaden.ibm.com is unable to contact the next router in the TCP/IP network.

MAXSOCKETS SIT parameter

The SIT parameter MAXSOCKETS controls the maximum number of sockets that may be managed in a CICS region. This is set as follows:

- If the user ID that CICS is running under has superuser authority, then the default value is 65535.
- If not, the default is the value of the MAXFILEPROC parameter specified in SYS1.PARMLIB member BPXPRMxx.

Note that sockets created by Java programs running on threads that are not managed by CICS do not count towards the limit.

The MAXSOCKETS value can be seen using the CICS command `CEMT I TCPIP` (see Figure 5-9). The MAX field shows the maximum number of sockets, and the ACT field shows how many sockets are active on the region. The value of MAXSOCKETS can also be changed using `CEMT`.

```
I TCPIP
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Tcp Ope Act(00006) Max( 00255 )











                                                 SYSID=PJA1 APPLID=SCSCPJA1
```

Figure 5-9 CEMT showing MAXSOCKETS

CICS statistics

It is possible to obtain statistics from CICS about TCP/IP and ECI over TCP/IP. This is done with the CICS Statistics Print sample program. To use this, we performed the following steps:

1. We installed the group DFH$STAT with the CICS command `CEDA INS G(DFH$STAT)`.
2. We ran the statistics print transaction with the CICS command `STAT`.
3. We pressed PF4. This displayed the Report Selection screen.
4. We selected the CICS functions TCP/IP and TCP/IP Services, as shown in Figure 5-10 on page 95.

```
          Sample Program - CICS Statistics Print Report Selection
                                                     07/03/2002  15:00:08

Select the statistics reports required and press 'Enter' to validate

 System Status. . . . . . . . . . . Y Page Index . . . . . . . . . . . . N
 Transaction Manager. . . . . . . . N Dispatcher . . . . . . . . . . . . N
 Storage Manager. . . . . . . . . . N
 Loader . . . . . . . . . . . . . . N Storage Subpools . . . . . . . . . N
 Transactions . . . . . . . . . . . N Transaction Classes. . . . . . . . N
 Programs . . . . . . . . . . . . . N
 Programs by DSA and LPA. . . . . . N DFHRPL Analysis. . . . . . . . . . N
 Transient Data . . . . . . . . . . N Transient Data Queues. . . . . . . N
 Temporary Storage. . . . . . . . . N Temporary Storage Models . . . . . N
 Temporary Storage Queues . . . . . N Temporary Storage Queues by Pool . N
                                      Logstream Global (System Logs) . . N
 Journals . . . . . . . . . . . . . N Logstreams . . . . . . . . . . . . N
 Files. . . . . . . . . . . . . . . N Data Set Names . . . . . . . . . . N
 LSR Pools. . . . . . . . . . . . . N Coupling Fcty Data Table Pools . . N

 TCP/IP . . . . . . . . . . . . . . Y TCP/IP Services. . . . . . . . . . Y
 Report selections are valid.
 Return to the Statistics Print screen to print the selected reports.
 F1=Help    F3=Return to Print    F8=Forward      F10=Save     F12=Restore
```

Figure 5-10 STAT report selection screen

5. We pressed PF3. The main panel was displayed. We pressed PF5 to print the statistics to the CICS log.

The DDNAME of the statistics print output under SDSF was S0000001. A summary of the statistics output is shown in Example 5-10.

Example 5-10 Partial CICS statistics

```
System Status
_____________

Max IP Sockets. . . . . . . . . . :     255
Active IP Sockets.. . . . . . . . :       6

TCP/IP Services
_______________

TCP/IP    Service      Port
Service   Status      Number  Protocol  Backlog  IP Address
___________________________________________________________
```

```
PJA1      OPEN         8010     IIOP          5  9.12.6.29
PJA1RMEB  OPEN         8011     HTTP          5  9.12.6.29
PJA1TCP   OPEN         8018     ECI         128  9.12.6.29

TCP/IP                            Attach
Service   SSL     Authenticate  Security  Tran  URM
___________________________________________________________
PJA1      None    None                    CIRR
PJA1RMEB  None    None                    CWXN  DFHWBADX
PJA1TCP   None    None            Local   CIEP

TCP/IP Services - Requests
__________________________

TCP/IP    Service        Open        Open     <- Connections ->
Service   Status         Date        Time     Current       Peak
______________________________________________________________
PJA1      OPEN        07/03/2002  13:31:07          0          0
PJA1RMEB  OPEN        07/03/2002  13:31:07          0          0
PJA1TCP   OPEN        07/03/2002  13:31:08          2          2


TCP/IP       Trans        Send     Avg Bytes    Receive     Avg Bytes
Service    Attached     Requests    / Send     Requests    / Receive
_____________________________________________________________________
PJA1              0           0           0           0           0
PJA1RMEB          0           0           0           0           0
PJA1TCP           2           4          47          12          29
```

As shown, the maximum IP sockets is 255. Details of the TCP/IP listener are given, including port number, backlog, attach security, and transaction. Details of requests made to our TCP/IP listener are also given, including when the listener was open, bytes sent, and requests received.

Common errors

If you experience problems connecting to CICS, check for the following message in the CICS log:

```
DFHIE1203 07/03/2002 14:20:20 SCSCPJA1 9.1.39.10 PJA1TCP EPI request
attempted by TCP/IP connected client.
```

The corresponding message in the Client daemon log is:

```
07/03/02 11:18:55.268 [3113] CCL:CCL3102 Inbound GDS data error (000C, 0,
12)
```

This occurs if an EPI request is flowed to the TCP/IP listener, for example by trying to use **`cicsterm`** against the CICS region, since CICS TS V2.2 only supports ECI over TCP/IP.

5.4.2 Tracing

For further details on how to use the trace facilities provided with the Client daemon, refer to 3.5.2, "Tracing" on page 61.

CICS trace

The CICS IE domain provides ECI specific services and also the following:

- New trace levels IE = 1 or 2
- New dump parameters IE = 1, 2 or 3

To enable or change IE tracing, use the CICS-supplied transaction `CETR`.

eNetwork Communication Server TCP/IP Packet Trace

The eNetwork Communication Server TCP/IP Packet Trace can be used to trace the data being flowed into and out of a CICS region, for data conversion or other diagnosis. For information about TCP/IP Packet Trace, refer to *OS/390 eNetwork Communication Server IP Diagnosis Guide*, SC31-8521.

6

CICS TG security scenarios

In this chapter, we describe how to use and implement CICS-RACF security when using the CICS Transaction Gateway (CICS TG).

We start with a brief overview of CICS security and then move on to document a set of end-to-end security scenarios to show you how we secured the following scenarios:

- ECI request to CICS using the CICS TG for z/OS
- ECI request to CICS using the CICS TG for Windows
- EPI request to CICS using the CICS TG for Windows

6.1 Introduction to CICS security

CICS uses the z/OS System Authorization Facility (SAF) to route authorization requests to an external security manager (ESM) to perform all its security checks. Any suitable ESM could be used, but as IBM's RACF product is the most commonly used, the remainder of this book will refer to RACF. For complete information about CICS security, refer to the *CICS RACF Security Guide*, SC33-1701.

Every CICS region requires certain special user IDs to be established, and also uses certain user ID when receiving inbound requests from other systems. These user IDs are as follows:

Region user ID This is the user ID under which the CICS job itself runs, and is a powerful user ID.

Default user ID This is used when users do not explicitly sign on, and should be given very low authorization. It is specified in the SIT parameter DFLTUSER.

Flowed user ID This is any user ID that is flowed in an ISC or MRO request, and includes user IDs flowed in ECI and EPI requests from Java applications.

Link user ID This is a user ID defined on CONNECTION or SESSIONS definition. It is used in link security and to determine if connected systems are *equivalent*.

Authentication of CICS users is the responsibility of RACF. Once authenticated, the user can pass through transaction security, resource security, command security, surrogate security, and, if the request is forwarded to another CICS region, intercommunication security. These are briefly explained in the following text.

Transaction security

CICS uses transaction security to control a user's permission to start a transaction. CICS performs a transaction security check even if no user has signed on. Users who do not sign on can use only those transactions that are authorized to the CICS *default user ID*. Usually this ID is very limited in what it has access to.

Resource security

CICS provides a further (optional) level of security by controlling access to individual resources, which include programs, files, and started transactions. There are no special implications for resource security with the CICS TG and so this subject is not addressed any further in this chapter.

Surrogate user security

CICS performs surrogate user security checking in a number of instances to ensure that the authenticated user is authorized to act for another user. It can also be used by the CICS TG itself to ensure that the started task user ID is authorized to initiate work on behalf of the user ID flowed in an ECI request.

Intercommunication security

Intercommunication security in CICS is concerned with incoming requests for access to CICS resources. Requests from the CICS TG can arrive via APPC (ISC) or EXCI (MRO) connections and the two are treated somewhat differently.

There are three fundamentally different intercommunication security checks that can be performed as follows:

- *Bind security*

  This verifies the *system* wishing to connect (bind) to CICS is authorized to do so.

- *User security* (or in LU 6.2 terms, *conversation level security*)

  This causes a check to be made against the flowed user ID when an inbound requests attaches the requested transaction in CICS. This behavior is defined in the ATTACHSEC parameter on the CONNECTION definition. For MRO or EXCI requests from the CICS TG, this should always be IDENTIFY, meaning that only the user ID is checked. For APPC (or TCP62) connections from the CICS TG, this should be set to VERIFY, meaning both the user ID and password are checked. For ECI over TCP/IP connection from the CICS TG, this should be set in the TCPIPSERVICE to VERIFY.

- *Link security*

  This is an additional level of authorization checking that can apply to the attached transaction. A specific user ID (the *link user*) is defined on the connection with the remote system. This user ID must be authorized to have access to all transactions and resources invoked through this connection. This concept applies equally to MRO and ISC.

We discuss how each of these apply to security with CICS TG in this chapter.

6.2 CICS TG security scenarios

In the following sections, we present three security scenarios, as follows:

- An ECI call to a CICS program using the CICS TG for z/OS and an EXCI connection to CICS (see 6.2.1, “ECI to CICS TG for z/OS (EXCI)” on page 102)

- An ECI call to a CICS program using the CICS TG for Windows and a TCP/IP connection to CICS (see 6.2.2, “ECI to CICS TG for Windows (TCP/IP)” on page 112)
- An EPI call to a CICS transaction using the CICS TG for Windows and a TCP62 connection to CICS (see 6.2.3, “EPI to CICS TG for Windows (TCP62)” on page 115)

The aim of each scenario is to allow the user CICSRS2 access to the desired resource but deny the user CICSRS5 access to the same resource.

In each example we show you how we configured all the required security definitions in both the CICS TG and CICS. We do not detail the basic setup of the CICS TG, since this is detailed in other chapters in the book. To test each scenario, we use a simple Java test application, either using the supplied samples (EciB1) or our own samples (SignonCapable and SignonIncapable), which are supplied with the additional material for this book (see Appendix C, “Additional material” on page 389).

6.2.1 ECI to CICS TG for z/OS (EXCI)

In this example, we show how to secure a CICS program that is called from an ECI application using the CICS TG for z/OS (Figure 6-1). To test the scenario we used the synchronous ECI sample EciB1 sample provided by the CICS TG in CTGSAMPLES.JAR. The platform of the Java client in our scenario is not important for our test purposes and could have equally been any other platform (such as UNIX System Services or Linux) used in this book.

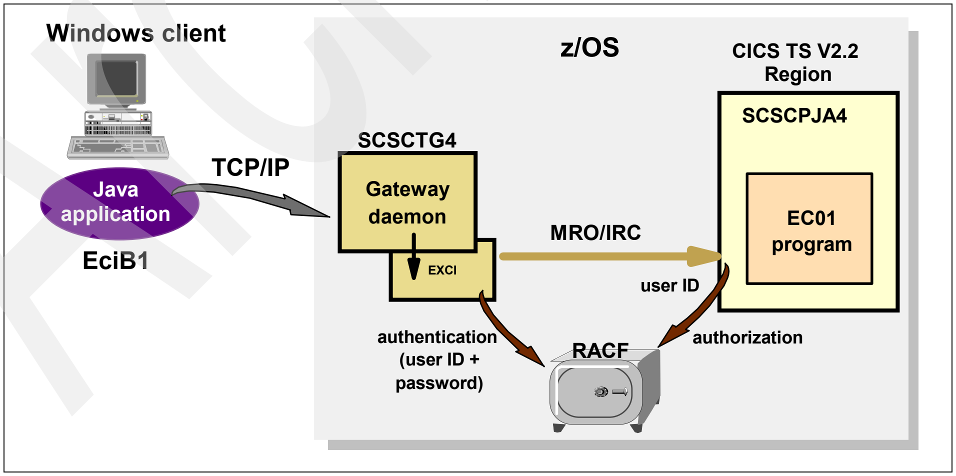


Figure 6-1 CICS TG for z/OS security test environment

In Table 6-1 we document the user ID and job names used in our configuration.

Table 6-1 Configuration parameters

CICS APPLID	SCSCPJA4
CICS default user ID (DFLTUSER)	CICSUSER
CICS region user ID	SCSCPJA4
Gateway daemon started task	SCSCTG4
Started task user ID	CTGUSER
TCP/IP hostname	wtsc66oe.itso.ibm.com
CICS TG TCP/IP port	2007
Flowed user ID to which we wish to *permit* access	CICSRS2
Flowed user ID to which we wish to *deny* access	CICSRS5
CICS program to be invoked	EC01

Note: The TCP/IP port 2007 is not the default port for the TCP protocol handler. We already have another Gateway daemon (SCSCTG5) using the default port 2006, and so we are forced to use a different port for our secure Gateway daemon.

For more information on EXCI security, refer to the *CICS External Interfaces Guide*, SC34-6006.

CICS configuration

Our CICS region, SCSCPJA4, was configured with security prefixing and transaction security active using the following parameters:

- `IRCSTRT=YES`
- `SECPRFX=YES`
- `SEC=YES`
- `XDCT=NO`
- `XFCT=NO`
- `XPCT=NO`
- `XPPT=NO`
- `XTRAN=YES`
- `XCMD=NO`

Note: We used security prefixing (SECPRFX=YES) in our CICS region, which prevents our RACF security profiles from affecting other CICS regions. This can be quite useful in a production environment, since it means all security profiles are unique to an individual region, but conversely it can mean more work for the security administrator because more profiles must be defined.

Basic CICS TG configuration

The commands listed in the following sections are in addition to the basic configuration necessary for normal functioning of the CICS TG for z/OS. Before you implement any security in your environment, we recommend that you set up and test a non-secure environment as documented in Chapter 4, "EXCI connections to CICS" on page 65 and Chapter 7, "TCP connections to the Gateway daemon on z/OS" on page 133. In this chapter, you will find documented the following actions, which are necessary for the normal functioning of the CICS TG in a non-secure environment:

- Setup of the started task user ID
- Access to the TCPIP.STANDARD.TCPXLBIN data set
- Removal of the share bit (s extended attribute) from the ctgstart script
- Access to the BPX.SERVER profile
- Enabling of program control for CICS TG data sets and HFS files

We now document the following steps necessary to secure access to our CICS program EC01:

- Configure MRO bind time security
- Enable CICS TG password checking
- Configure security for CICS CONNECTION and SESSIONS definitions
- Configure the flowed user ID
- Permit access to the mirror transaction, CSMI
- Define RACF surrogate profiles

Tip: We encountered problems when running a transactional EXCI request (using EciI1) with security enabled on the CICS TG. Each time we ran a request we would see *dirty address space* errors when attempting to access module DFHXCSVC in the SDHFLINK library. Marking CICSTS22.CICS.SDFHLINK as program controlled solved this problem.

MRO bind security (DFHAPPL FACILITY class profiles)

MRO bind security prevents unauthorized attached MRO regions from starting transactions in a CICS region, and as such applies equally to the CICS TG, as a user of MRO, as it does to CICS. It is implemented using two different DFHAPPL FACILITY class profiles that control logon to IRP. To log on to IRP, the user ID under which the CICS TG runs requires the following permissions:

- Update access to the RACF FACILITY class DFHAPPL.*DFHJVPIPE*, where *DFHJVPIPE* is the EXCI pipe name as defined in the CICS TG environment variable DFHJVPIPE, and in the NETNAME parameter in the CICS SESSIONS definition. If a generic EXCI connection is used, there is no pipe name and this does not apply.
- Read access to the RACF FACILITY CLASS DFHAPPL.*APPLID*, where *APPLID* is the APPLID of the CICS region in question.

> **Note:** Use of MRO bind-time security is optional and if these DFHAPPL profiles are *not* defined, then any MRO connected system will be able to connect to your CICS region.

We activated MRO bind security for our configuration as follows:

1. We gave update access to the RACF FACILITY class DFHAPPL.*DFHJVPIPE* using the following RACF command:

```
RDEFINE FACILITY (DFHAPPL.SCSCTG4) UACC(NONE)
PERMIT DFHAPPL.SCSCTG4 CLASS(FACILITY) ID(CTGUSER) ACCESS(UPDATE)
SETROPTS RACLIST(FACILITY) REFRESH
```

> **Note:** We defined all of our security profiles with a UACC(NONE), meaning a universal access of none. Thus by default all users were denied access, and permissions were then granted for each user ID as required.

2. We gave read access to the RACF FACILITY class DFHAPPL.*APPLID* with the following RACF command:

```
RDEFINE FACILITY (DFHAPPL.SCSCPJA4) UACC(NONE)
PERMIT DFHAPPL.SCSCPJA4 CLASS(FACILITY) ID(CTGUSER) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

> **Tip:** The `SETROPTS` command above must be run every time you modify any RACF profiles, and will cause all the profiles in the specified RACF class (in our case FACILITY) to be activated in the LPAR.

If your CICS TG for z/OS fails an MRO bind security check, you will receive the error shown in Example 6-1 on page 106 in the CICS TG JNI trace. Details on how to activate the JNI tracing can be found in “JNI trace” on page 176 in Chapter 7.

Example 6-1 JNI trace of an MRO bind security failure

```
EXCI function error.  Function Call = 6, Response = 12, Reason = 414,
Subreason field-1 = 0, subreason field-2 = 0, Cics_Rc = -9
Error message from CICS: DFHAC2033 .
```

Debugging errors shown in the JNI trace is explained further in 7.4, "Problem determination" on page 164. However, the CICS error message DFHAC2033 should be familiar to most CICS systems programmers.

CICS TG password checking

To enable the CICS TG to authenticate each user ID and password flowed on an ECI request, the environment variable AUTH_USERID_PASSWORD must be set in the CICS TG environment variables. We set this variable in our CTG4ENV PDS member (Example 6-2), which we supplied as input to our CICS TG started task, using the STDENV member. See 7.2.1, "Defining CICS TG environmental variables" on page 148 for more information on the STDENV DD.

Example 6-2 CICS TG environment variables

```
DFHJVPIPE=SCSCTG4
AUTH_USERID_PASSWORD=YES
JAVA_PROPAGATE=NO
CTG_RRMNAME=CCL.CTG.IBM.UA
CICSCLI=/ctg/scsctg4/CTG.INI
CTGSTART_HOME=/usr/lpp/ctg402/ctg/bin
EXCI_LOADLIB=CICSTS22.CICS.SDFHEXCI
DFHJVSYSTEM_01=SCSCPJA1 - CICSTS 1.3
DFHJVSYSTEM_02=SCSCPJA4 - CICSTS 2.2
```

CONNECTION and SESSIONS definitions

In order for our CICS region to use the user ID flowed in the EXCI call from the CICS TG, we set the parameter ATTACHSEC=IDENTIFY in our EXCI connection definition in our CICS region SCSCPJA4, as shown in Figure 6-2.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0620
 CEDA  ALter CONnection( CTG4 )
  CONnection     : CTG4
  Group          : PJA4CTG4
  DEscription  ==>
 CONNECTION IDENTIFIERS
  Netname      ==> SCSCTG4
  INDsys       ==>
 CONNECTION PROPERTIES
  ACcessmethod ==> IRc                 Vtam | IRc | INdirect | Xm
  PRotocol     ==> Exci                Appc | Lu61 | Exci
  Conntype     ==> Specific            Generic | Specific
 SECURITY
  SEcurityname ==>
  ATtachsec    ==> Identify            Local | Identify | Verify
                                       | Persistent | Mixidpe
  BINDPassword   :                     PASSWORD NOT SPECIFIED
  BINDSecurity ==> No                  No | Yes
  Usedfltuser  ==> No                  No | Yes

                                                   SYSID=PJA4 APPLID=SCSCPJA4
```

Figure 6-2 CICS CONNECTION definition

IDENTIFY means that CICS uses the flowed user ID in the EXCI request, but does not expect a password to be flowed with the request, as this should be checked by the CICS TG itself.

Link security

Next we decided to disable link security. Link security is an additional level of security that applies to all attach requests received over a connection. For MRO or EXCI requests, this is set as follows.

The SESSIONS definition is checked as follows:

1. If the link user ID is the same as the region user ID, then the systems are deemed equivalent and no link security authorization is performed.
2. If the link user ID is defined as anything else, then this user ID must have access to all resources that the EXCI requests need.

3. If the USERID is blank, then the user ID of the connected region (that is, the user ID under which the CICS TG runs) is the link user ID.

To disable link security means that we set our CICS TG region, SCSCTG4, and our CICS region, SCSCPJA4, to be *equivalent systems*. To do this, we set the USERID parameter in the SESSIONS definition to be SCSCPJA4, which is the CICS region user ID. Our SESSIONS definition is shown in Figure 6-3.

```
OVERTYPE TO MODIFY                                         CICS RELEASE = 0620
 CEDA  ALter Sessions( CTG4SESS )
  Sessions       : CTG4SESS
  Group          : PJA4CTG4
  DEscription  ==>
 SESSION IDENTIFIERS
  Connection   ==> CTG4
  SESSName     ==>
  NETnameq     ==>
  MOdename     ==>
 SESSION PROPERTIES
  Protocol     ==> Exci                Appc | Lu61 | Exci
  MAximum      ==> 000 , 000           0-999
  RECEIVEPfx   ==> C4
  RECEIVECount ==> 010                 1-999
 PRESET SECURITY
  USERId       ==> SCSCPJA4
                                                   SYSID=PJA4 APPLID=SCSCPJA4
```

Figure 6-3 CICS SESSIONS definition

Many other intercommunication security configurations are possible. Table 6-3 on page 120 lists the different settings for link security and ATTACHSEC and how they interoperate. In a production system, it is recommended that you specify ATTACHSEC=IDENTIFY and use non-equivalent systems, so that a link user ID is also used.

Table 6-2 Attach security settings with an EXCI connection from the CICS TG

Equivalent systems?	**Link user ID = region user ID**		**Link user ID not = region user ID**	
ATTACHSEC	**LOCAL**	**IDENTIFY**	**LOCAL**	**IDENTIFY**
Link user ID check	NO	NO	YES	YES
Flowed user ID check	NO	YES	NO	YES
User ID mirror transaction runs under in CICS	CICS default user ID	Flowed user ID	Link user ID	Flowed user ID

Configure the flowed user ID

Because the CICS TG runs as a shell process under UNIX System Services, any user ID that it tries to authenticate with RACF, such as a user ID flowed with an ECI request, must have an OMVS segment defined in its RACF profile. See *OS/390 UNIX System Services Planning,* GA22-7800 for more information.

Access to the mirror transaction, CSMI

An EXCI request received by CICS from the CICS TG for z/OS attaches a mirror transaction, by default CSMI. Therefore to authorize only our user ID CICSRS2 to have access to the mirror transaction, we issued the following RACF command to define read access to the mirror transaction in the CLASS TCICSTRN:

```
RDEF TCICSTRN SCSCPJA4.CSMI UACC(NONE)
PERMIT SCSCPJA4.CSMI CL(TCICSTRN) ID(CICSRS2) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

Tip: For our examples, we permit access to a single user (CICSRS2). In a production environment, you will probably create a group of users requiring common access. Once a group is built, you will permit access to a group. This permits user's access to be controlled by the group to which they belong, rather than by individual permissions. This can be used to simplify the security definitions required.

SURROGAT security profiles

In order for the EXCI to be able to switch the security context of the EXCI request to the flowed user ID, the correct SURROGAT security profiles must be defined. The user ID of the EXCI job (in our case the CICS TG started task) requires read access to the *USERID*.DFHEXCI SURROGAT class profile, where *USERID* is the flowed user ID in the EXCI request.

We issued the following commands to permit our CICS TG started task user ID (CTGUSER) to switch to the user ID CICSRS2 that we wish to flow in the ECI request:

```
RDEFINE SURROGAT CICSRS2.DFHEXCI UACC(NONE) OWNER(CICSRS2)
PERMIT CICSRS2.DFHEXCI CLASS(SURROGAT) ID(CTGUSER) ACCESS(READ)
SETROPTS RACLIST(SURROGAT) REFRESH
```

Note: It is also possible to disable surrogate security by either reassembling the EXCI options table DFHXCOPT, with SURROGAT=NO, or by using a RACF surrogate profile with universal READ access such as:

```
RDEFINE SURROGAT *.DFHEXCI UACC(READ)OWNER (CICSRS2)
```

Testing

In order to test our secure CICS TG for z/OS environment, we chose to use the EciB1 synchronous ECI sample. The runtime class for this is provided by the CICS TG in the CTGSAMPLES.JAR and the source is also provided in the samples\java\com\ibm\ctg\samples\eci directory. To test using the version in the CTGSAMPLES.JAR, it is merely necessary to copy CTGLCLIENT.JAR and CTGSAMPLES.JAR to your desired machine and set your classpath to contain these two JAR files.

We ran our tests from a Windows workstation with the CICS TG already installed, so we set our classpath using the following command:

```
SET CLASSPATH=C:\Program Files\IBM\IBM CICS Transaction Gateway\
Classes\CTGCLIENT.JAR;C:\Program Files\IBM\IBM CICS Transaction Gateway\
Classes\CTGSAMPLES.JAR
```

Example 6-3 shows the results of testing EciB1 from a Windows workstation.

Example 6-3 Testing z/OS EXCI security with EciB1 sample

```
C:\>java com.ibm.ctg.samples.eci.EciB1 tcp://wtsc66oe.itso.ibm.com 2007

CICS TG Basic ECI Sample 1
Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway Url]
                                          [Gateway Port Number]

To enable client tracing, run the sample with the following Java option:
 -Dgateway.T.trace=on
The address of the Gateway has been set to TCP://WTSC66OE.ITSO.IBM.COM
Port:2007

CICS Servers Defined:
         1. SCSCPJA1 -
         2. SCSCPJA4 -

Choose Server to connect to, or q to quit:
2

Enter your CICS User ID:
CICSRS2

Enter your CICS Password:
PASSWORD

Program EC01 returned with data:-
        Hex: 30362f30362f30322031343a31383a30380
        ASCII text: 06/06/02 21:19:08
```

EciB1 is written so that it will initially make a call to the specified CICS region without a user ID and password, and then if it receives a security error it will then prompt you for a user ID and password. It also allows trace to be activated by specifying `-Dgateway.T.trace=on`. This will trace the CICS TG calls made in the Java application and is particularly useful in showing the user ID and COMMAREA flowed to and from the CICS TG.

When we ran EciB1 with trace on, we saw the following input on the first ECI call to the CICS TG specifying SCSCPJA4 as the CICS region. As seen in Example 6-4, the user ID at offset 57 is not set (low values), because we have not been prompted for it as yet.

Example 6-4 EciB1 object/COMMAREA on initial call (created with -Dgateway.T.trace=on)

```
S-C: CCL6603I: # 00000: 47 61 74 65 00 40 00 00 00 00 00 01 00 00 00 00   Gate.@..........
S-C: CCL6603I: # 00016: 00 00 00 00 00 00 00 03 45 43 49 00 00 00 00 60   ........ECI....`
S-C: CCL6603I: # 00032: 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00   ................
S-C: CCL6603I: # 00048: 00 53 43 53 43 50 4A 41 34 00 00 00 00 00 00 00   .SCSCPJA4.......
S-C: CCL6603I: # 00064: 00 00 00 00 00 00 00 00 00 2A 2A 2A 2A 2A 2A 2A   .........*******
```

In Example 6-5, we see the second ECI request to the CICS TG, after we had been prompted for a user ID and password. This time the user ID (CICSRS2) is contained in the ECI request and you can see the password (********) is masked for security reasons.

Example 6-5 EciB1 object/COMMAREA after ID and password input (created with -Dgateway.T.trace=on)

```
S-C: CCL6603I: # 00000: 47 61 74 65 00 40 00 00 00 00 00 01 00 00 00 00   Gate.@..........
S-C: CCL6603I: # 00016: 00 00 00 00 00 00 00 03 45 43 49 00 00 00 00 60   ........ECI....`
S-C: CCL6603I: # 00032: 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00   ................
S-C: CCL6603I: # 00048: 00 53 43 53 43 50 4A 41 34 63 69 63 73 72 73 32   .SCSCPJA4CICSRS2
S-C: CCL6603I: # 00064: 00 00 00 00 00 00 00 00 00 2A 2A 2A 2A 2A 2A 2A   .........*******
```

6.2.2 ECI to CICS TG for Windows (TCP/IP)

In this example we show you how we configured security to allow an ECI-based Java application to make an authenticated call to CICS using the facilities of the CICS TG for Windows, using a TCP/IP connection to our CICS TS V2.2 region (Figure 6-4).

As in the previous example, we used the supplied sample EciB1, and the CICS server program, EC01, in our test. The aim of our test was to permit the user ID CICSRS2 access to the EC01 program and deny access to the user CICSRS5.

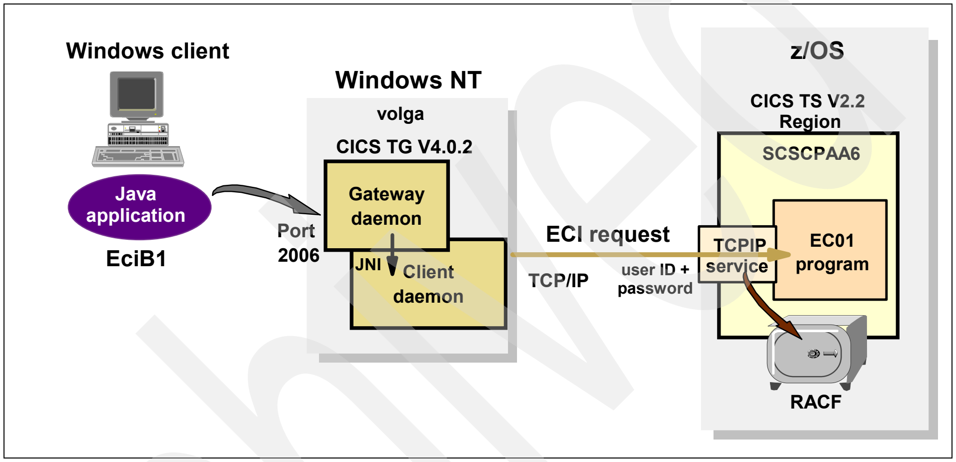


Figure 6-4 ECI to CICS TG for Windows, security scenario

TCP/IP security configuration

When using TCP/IP connections to CICS, the CICS TG provides support for the flowing of a user ID and password within the ECI request, for authentication of this user ID and password within CICS, and authorization of requests using RACF. In the instructions that follow, we detail how we configured security in our CICS region and CICS TG.

Basic CICS TG configuration

In this section, we assume that you have already installed the CICS TG for Windows and configured a TCP/IP connection from the CICS TG to CICS TS V2.2. For further details on how to configure TCP/IP connections, refer to Chapter 5, “TCP/IP connections to CICS” on page 81.

CICS region configuration

We used the same secure CICS region, SCSCPJA4, as we used in our previous tests with the CICS TG for z/OS. For further details on the security settings and SIT parameters, refer to "CICS configuration" on page 103.

CICS TCPIPSERVICE definition

In order for our CICS region to use the user ID and password flowed in the ECI request from the CICS TG, it is necessary to activate security in the TCP/IP service definition. To do this, we defined a TCPIPSERVICE definition and set ATTACHSEC=VERIFY, as shown in Figure 6-5. The port we used for this TCPIPSERVICE definition was 8048, meaning that the CICS region listens for ECI requests on this TCP/IP port. This port that must match the Port parameter defined in the CICS TG Configuration Tool TCP/IP settings (for details on how we defined this, see page 87 in Chapter 5).

```
OVERTYPE TO MODIFY                                              CICS RELEASE = 0620
 CEDA  ALter TCpipservice( PJA4TCP  )
  TCpipservice   : PJA4TCP
  GROup          : PJA4GRP
  DEscription  ==>
  Urm          ==>
  POrtnumber   ==> 08048         1-65535
  STatus       ==> Open          Open | Closed
  PRotocol     ==> Eci           Iiop | Http | Eci
  TRansaction  ==> CIEP
  Backlog      ==> 00128         0-32767
  TSqprefix    ==>
  Ipaddress    ==>
  SOcketclose  ==> No            No | 0-240000 (HHMMSS)
 SECURITY
  SSl          ==> No            Yes | No | Clientauth
  Certificate  ==>
  AUthenticate ==>               No | Basic | Certificate | AUTORegister
                                 | AUTOMatic
  ATtachsec      : Verify        Local | Verify

                                                        SYSID=PJA4 APPLID=SCSCPJA4
```

Figure 6-5 CICS TCPSERVICE definition

Permit access to the mirror transaction

Any ECI requests received from the CICS TG for Windows run under a mirror transaction. By default, this will be the ASCII mirror, CPMI. To enable our desired user ID CICSRS2 to have access to this transaction, we issued the following RACF commands to grant access to the SCSCPJA4.CPMI profile to the user CICSRS2:

```
RDEF TCICSTRN SCSCPJA4.CPMI UACC(NONE)
PERMIT SCSCPJA4.CPMI CLASS(TCICSTRN) ID(CICSRS2) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

Testing

In order to test our secure CICS TG for Windows environment, we again used the EciB1 synchronous ECI sample. This runtime class for this is provided by the CICS TG in the CTGSAMPLES.JAR and the source is also provided in the samples\java\com\ibm\ctg\samples\eci directory. To test using the version in the CTGSAMPLES.JAR, it is merely necessary to copy CTGLCLIENT.JAR and CTGSAMPLES.JAR to your desired machine and set your classpath to contain these two JAR files.

We ran our tests from a Windows workstation with the CICS TG already installed, so we set our classpath using the following command:

```
SET CLASSPATH=C:\Program Files\IBM\IBM CICS Transaction Gateway\
Classes\CTGCLIENT.JAR;C:\Program Files\IBM\IBM CICS Transaction Gateway\
Classes\CTGSAMPLES.JAR
```

Since we have set up security so a user ID and password are required, EciB1 detects a security error caused by the first request, which includes a null user ID and password. EciB1 then prompts for a user ID and password, which it flows with the ECI request. Example 6-6 shows the results of a successful test with EciB1.

Example 6-6 Testing z/OS EXCI security with EciB1 sample

```
C:\>java com.ibm.ctg.samples.eci.EciB1 tcp://volga.almaden.ibm.com 2007

CICS Transaction Gateway Basic ECI Sample 1
-------------------------------------------
Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway Url]
                                          [Gateway Port Number]

To enable client tracing, run the sample with the following Java option:
 -Dgateway.T.trace=on
The address of the Gateway has been set to tcp://volga.almaden.ibm.com
Port:2007

CICS Servers Defined:
```

```
        1. SC62PJA1 -CICSTS 2.2 AT SC66
        2. SCSCPJA4 -CICSTS 2.2 AT SC66

Choose Server to connect to, or q to quit:
2

Enter your CICS User ID:
CICSRS2

Enter your CICS Password:
PASSWORD

Program EC01 returned with data:-
        Hex: 32332f30362f30322031303a35373a34370
        ASCII text: 23/06/02 10:57:47
```

6.2.3 EPI to CICS TG for Windows (TCP62)

In this example, we show you how we configured security to allow an EPI-based Java application to make an authenticated call to CICS using the facilities of the CICS TG for Windows, using a TCP62 connection to our CICS TS V2.2 region (Figure 6-6). Since we were using the EPI to access a 3270-based CICS transaction, we decided to use the TCP62 communication protocol, since EPI is not supported either by the CICS TCP/IP protocol or by the CICS TG for z/OS.

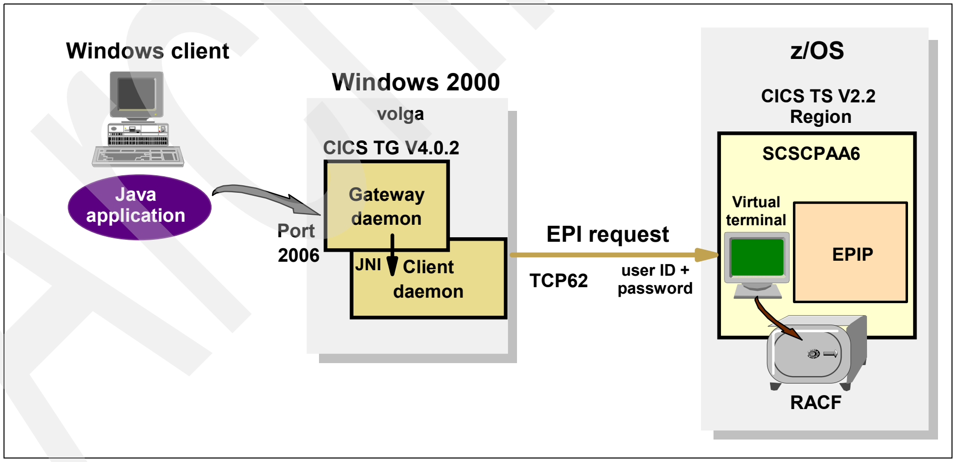


Figure 6-6 EPI to CICS TG for Windows, security scenario

Unlike the previous ECI examples, we decided to write our own test applications, because we wanted to show the difference between signon capable and signon incapable terminals. Our sample Java applications (SignonCapable and

SignonIncapable) are written to use the EPI support classes, and are available with the additional materials for this book (see Appendix C, "Additional material" on page 389). Both of these applications work with the CICS 3270 program, EPIPROG, which is a very simple BMS-based application that returns a map displaying the CICS region APPLID, date, time, and the user ID the task ran under (Figure 6-7).

```
 EPIPROG OUTPUT

APPLID:       SCSCPJA4
DATE:         25/06/02
TIME:         00:45:38
USERID:       CICSRS2
```

Figure 6-7 EPIP 3270 screen

The source code (EPIPROG), and the mapset (EPIPMAPS) for this application are provided in the additional material available with this redbook, along with the associated map class EPIMAPMap.java.

Signon capable terminals

When using the facilities of the EPI, the CICS TG creates a logical terminal in CICS for you, which is then used to run the 3270 transaction. For an EPI application to be able to start a secured CICS transaction on this terminal, it must supply security credentials (a user ID and password) for the CICS server to authenticate. There are two options available to do this:

- **Sign on to the CICS terminal**

  The security credentials determined are established at sign-on (by starting a transaction such as CESN). These credentials are then used for any subsequent authorization checks when starting other transactions. The user ID and password need not flow with further requests following the sign-on. This requires a signon capable EPI terminal.

- **Flow a user ID and password with each EPI request**

  A valid user ID and password must be flowed with each EPI request. This method does not require the user to sign on to CICS, and so uses a signon incapable EPI terminal.

Therefore, when writing an EPI application the application designer must choose between using a signon capable or signon incapable terminal, both of which require the use of an EPI *extended terminal*. For further details on our tests with signon capable and signon incapable terminals. refer to “EPI security choices” on page 120.

> **Note:** It is also possible to use a *basic terminal* in EPI applications. With a basic terminal, there is no ability to set the sign-on capability, and instead, any security credentials (the user ID and password) must be hardcoded on each server connection definition in the CICS TG configuration file. We do not discuss this option in this book, since the use of *extended terminals* allows an individual user ID and password to be flowed on each EPI request and therefore provides a more secure alternative. For further details on developing EPI applications, refer to the IBM redbook *Java Connectors for CICS*, SG24-6401.

CICS CONNECTION and SESSIONS definitions

In our example we decided to use connection autoinstall for our TCP62 connections. To configure security for these autoinstalled connection, we created a copy of the supplied APPC connection templates as follows:

```
CEDA COPY CONN(CBPS) TO GROUP(PJA4AI62)
CEDA COPY SESS(CBPS) TO GROUP(PJA4AI62)
```

We then modified the CONNECTION definition as shown in Figure 6-8 on page 118, ensuring that we specified the following parameters:

- ACCESSMETHOD(VTAM)
- PROTOCOL(APPC)

  These parameters are used to indicate that VTAM will be used with this APPC connection.

- ATTACHSEC(VERIFY)

  This is used to ensure that the autoinstalled CONNECTION will require the incoming user ID and password to be verified.

- USEDFLTUSER(YES)

  This parameter specifies that a terminal install request (using the CTIN transaction) without a user ID and password will run without a security check under the default CICS user ID.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0620
 CEDA  ALter CONnection( CBPS )
  CONnection     : CBPS
  Group          : PJA4AI62
  DEscription  ==> APPC AUTOINSTALL TEMPLATE, WITH SECURITY
 CONNECTION IDENTIFIERS
  Netname      ==> TMPLATE1
  INDsys       ==>
 CONNECTION PROPERTIES
  ACcessmethod ==> Vtam            Vtam | IRc | INdirect | Xm
  PRotocol     ==> Appc            Appc | Lu61 | Exci
  Conntype     ==>                 Generic | Specific
  SIngleseess  ==> No              No | Yes
  DAtastream   ==> User            User | 3270 | SCs | STrfield | Lms
  RECordformat ==> U               U | Vb
 SECURITY
  SEcurityname ==> SCSCPJA4
  ATtachsec    ==> Verify          Local | Identify | Verify | Persistent
                                   | Mixidpe
  BINDPassword   :                 PASSWORD NOT SPECIFIED
  BINDSecurity ==> No              No | Yes
  Usedfltuser  ==> No              No | Yes

                                                  SYSID=PJA4 APPLID=SCSCPJA4
```

Figure 6-8 CBPS CONNECTION definition

We also modified the SESSIONS definition as shown in Figure 6-9 on page 119 ensuring that we specified the following parameters:

- MODENAME(APPCMODE)
- MAXIMUM=(08,001)

  These parameters were used to control the APPC mode group name and properties as we had also defined in our TCP62 connection in the CICS TG.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0620
 CEDA  ALter Sessions( CBPS     )
  Sessions       : CBPS
  Group          : PJA4AI62
  DEscription  ==> APPC AUTOINSTALL TEMPLATE, WITH SECURITY
 SESSION IDENTIFIERS
  Connection   ==> CBPS
  SESSName     ==>
  NETnameq     ==>
  MOdename     ==> APPCMODE
 SESSION PROPERTIES
  Protocol     ==> Appc              Appc | Lu61 | Exci
  MAximum      ==> 008 , 001         0-999
  RECEIVEPfx   ==>
  RECEIVECount ==>                   1-999
  SENDPfx      ==>
  SENDCount    ==>                   1-999
  SENDSize     ==> 04096             1-30720
  RECEIVESize  ==> 04096             1-30720

                                                   SYSID=PJA4 APPLID=SCSCPJA4
```

Figure 6-9 CBPS SESSIONS definition

We then installed this CONNECTION template using the command:

```
CEDA INS GR(PJA4AI62)
```

Important: For an autoinstall connection to be successful, you must also set the autoinstall program to be DFHZATDY. For further details refer to 2.2.3, "CICS connection autoinstall" on page 23.

Link security

Next we decided to disable link security. Link security is an additional level of security that applies to all attach requests received over a connection. For APPC requests such as TCP62, this is set as follows.

The SESSIONS definition is checked as follows:

1. The SESSIONS definition is checked. If the USERID parameter is specified, this is used as the link user ID and no further checks are made.
2. The CONNECTION definition is checked, and the SECURITYNAME specified here is used as the link user ID.
3. If all else fails, the CICS default user ID will be used.

In disabling link security, we set our CICS TG region and our CICS region SCSCPJA4 to be equivalent systems. To achieve this, we set the SECURITYNAME parameter in the CONNECTION definition to be SCSCPJA4, which is the CICS region user ID. Our CONNECTION definition is shown in Figure 6-8 on page 118.

Many other intercommunication security configurations are possible. Table 6-2 on page 108 lists the different settings for link security and ATTACHSEC and how they interoperate. In a production system, it is recommended that you specify ATTACHSEC=IDENTIFY and use non-equivalent systems, so that a link user ID is also used for authorization. This can be used to limit the maximum authority any given user can obtain when using the connection.

Table 6-3 Attach security settings with a distributed CICS TG

Equivalent systems?	**Link user ID = region user ID**		**Link user ID not = region user ID**	
ATTACHSEC	**LOCAL**	**VERIFY**	**LOCAL**	**VERIFY**
Link user ID check	NO	NO	YES	YES
Flowed user ID check	NO	YES	NO	YES
User ID CICS transaction runs under	CICS default user ID	Flowed user ID	Link user ID	Flowed user ID

Securing access to the EPIP transaction

To secure the CICS transaction EPIP, we first defined the RACF profile SCSCPJA4.EPIP with a default access of NONE. This had the effect of denying access to all users. Then we permitted access to CICSRS2. The following commands were used to define the RACF profiles:

```
RDEF TCICSTRN SCSCPJA4.EPIP UACC(NONE)
PERMIT SCSCPJA4.EPIP CLASS(TCICSTRN) ID(CICSRS2) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

EPI security choices

The EPI provides support for two distinct methods for connecting to secured transaction in CICS. The two options available are:

- Sign on to the CICS terminal before sending an EPI request, or
- Flow a user ID and password with each request

We discuss each of these options in the following two scenarios.

Sign on to the CICS terminal

This requires the EPI application to invoke a sign-on transaction such as CESN. With this method, a valid user ID and password must be supplied at the time of the sign-on, but after this all transactions run on the terminal under these credentials. This requires the use of an EPI *signon capable* terminal, a code example for which is shown in Figure 6-10. The full version of this code is provided in our sample Java application SignonCapable.java in the `itso.cics.epi` package supplied with this book.

```
Terminal term =
    new Terminal(
        epiGate,                      // CICS TG URL
        cicsRegion,                   // CICS server
        null,                         // device type
        null,                         // netname
        Terminal.EPI_SIGNON_CAPABLE,  // signon flag
        termUserid,                   // userid for CTIN
        termPassword,                 // password
        0,                            // timeout
        null);                        // encoding

    term.connect();                   // install terminal with CTIN

Screen scr = term.getScreen();
scr.field(1).setText("CESN");         // set transaction to CESN
scr.setAID(AID.enter);
term.send();                          // start CESN transaction
......
scr.field(1).setText("EPIP");         // set EPIP user transaction
scr.setAID(AID.enter);
term.send();                          // start EPIP
```

Figure 6-10 Using a signon capable terminal

Verifying EPI signon capable security

In our scenario, we set USEDFLTUSER(YES) on our CONNECTION definition. This means the CTIN and CESN requests will run using the default user ID. Therefore we needed to supply only a valid user ID and password for the request to run our actual EPI transaction.

> **Tip:** If you set USEDFLTUSER(YES) on your CONNECTION definition, then you should realize that this will allow any user to install EPI terminals in your CICS region without any authentication, since any user ID and password specified on the `Terminal` constructor will be ignored for the install of the terminal by the CTIN transaction. If you wish to restrict this behavior, you should specify USEDFLTUSER(NO) on the CICS CONNECTION definition and then specify a valid user ID and password for the CTIN install. In Figure 6-10 this is controlled by the `termUserid` and `termPassword` fields. If you do this, you must also permit READ access to the CTIN transaction for the desired user ID using the following commands:
>
> ```
> RDEF TCICSTRN SCSCPJA4.CTIN UACC(NONE)
> PERMIT SCSCPJA4.CTIN CLASS(TCICSTRN) ID(CICSRS2) ACCESS(READ)
> SETROPTS RACLIST(TCICSTRN) REFRESH
> ```

In Example 6-7, you can see the results of a successful call to the EPIP transaction from our SignonCapable EPI application.

Example 6-7 Successful test with SignonCapable EPI application

```
C:\itsoctgv5\Java>java itso.cics.epi.SignonCapable tcp:\\volga SC62PJA4 CICSRS2
PASSWORD
Gateway URL: tcp://volga/
Region: SC62PJA4
Userid: CICSRS2
Password: PASSWORD

Adding signon capable terminal with userid:null
Netname: CCLIE017
Starting CESN
Entering userid and password
 DFHCE3549:Now signed on to CICS
Starting EPIP
EPIP results
 Time : 02:12:01
 Date : 25/06/02
 Applid : SCSCPJA4
 Userid : CICSRS2
Closing terminal and gateway
```

In contrast, Example 6-8 shows the result of an unauthorized user (CICSRS5) attempting to access the EPIP transaction.

Example 6-8 Security failure with SignonCapable EPI test

```
C:\itsoctgv5\Java>java itso.cics.epi.SignonCapable tcp:\\volga SC62PJA4 CICSRS5
PASSWORD
Gateway URL: tcp://volga/
Region: SC62PJA4
Userid: CICSRS5
Password: PASSWORD
Unknown EPI error encountered
Error message was :Map is not valid
com.ibm.ctg.epi.EPIMapException: Map is not valid
   at itso.cics.epi.EPIMAPMap.<init>(EPIMAPMap.java:42)
   at itso.cics.epi.SignonCapable.main(SignonCapable.java:159)
```

The `EPIMapException` error in Example 6-8 merely states that the screen received when running the EPIP transaction was not expected. This is because we used the EPI `Map` class to handle the expected output from the EPIP transaction. The actual cause of the error can be seen by examining the CICS JESMSGLG where the following security violation was logged.

Example 6-9 JES SYSLOG error for EPIP security violation

```
ICH408I USER(CICSRS5 ) GROUP(SYS1    ) NAME(CICS RESIDENT       )  552
  SCSCPJA4.EPIP CL(TCICSTRN)
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```

Flow a user ID and password with each EPI request

If your CICS application does not need to invoke an EXEC CICS SIGNON, then a better option is to use *signon incapable* terminals. With this method the EPI user is not required to explicitly sign on to CICS. Instead a user ID and password must be supplied with each EPI request, and these are flowed in the FMH attach header and verified for each request. A sample code snippet demonstrating this technique is shown in Figure 6-11 on page 124. The full version of this code is provided in our sample Java application SignonIncapable.java in the `itso.cics.epi` package supplied with this book.

```
new Terminal(
        epiGate,                      // CICS TG URL
        cicsRegion,                   // CICS server
        null,                         // device type
        null,                         // netname
        Terminal.EPI_SIGNON_INCAPABLE,
        userid,                       // userid for CTIN
        password,                     // password
        0,                            // timeout
        null);                        // encoding

Screen scr = term.getScreen();
term.connect();                       // install terminal with CTIN

term.setUserid(userid);               // set userid for EPI request
term.setPassword(password);           // set passowrd

scr.field(1).setText("EPIP");         // set EPIP user transaction code
scr.setAID(AID.enter);
term.send();                          // start EPIP transaction
....
```

Figure 6-11 Using a signon incapable terminal

To enable access from our SignonIncapable application to our EPIP test transaction, it was necessary to grant READ access for our CICSRS2 user ID to the RACF profiles protecting both the EPIP and CTIN transactions as follows:

```
PERMIT SCSCPJA4.CTIN CLASS(TCICSTRN) ID(CICSUSER) ACCESS(READ)
PERMIT SCSCPJA4.EPIP CLASS(TCICSTRN) ID(CICSRS2) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

Tip: Unlike signon capable terminals, we found that when using signon incapable EPI terminals, we also had to ensure the link user ID had READ access to the TCICSTRN profile protecting the EPIP and CTIN transactions. However, instead of creating another profile, we fixed this by making our systems equivalent, by setting the SECURITYNAME in the CONNECTION definition to be the same as the region user ID (SCSCPJA4), as shown in Figure 6-8.

Verifying EPI signon incapable security

In Example 6-10, we show a successful call to EPIP using our signon incapable EPI application.

Example 6-10 Successful test with signon incapable application

```
C:\itsoctgv5\Java>java itso.cics.epi.SignonIncapable tcp:\\volga SC62PJA4
CICSRS2 PASSWORD
Gateway URL: tcp://volga/
Region: SC62PJA4
Userid: CICSRS2
Password: PASSWORD

Adding signon incapable terminal
Netname: VOLGO015
Starting EPIP
 Time : 18:02:40
 Date : 26/06/02
 Applid : SCSCPJA4
 Userid : CICSRS2
Closing terminal and gateway
```

Example 6-11 shows the result of an unauthorized user attempting to access our example. The JES SYSLOG (Example 6-12) shows CICSRS5 as failing in an attempt to access CTIN.

Example 6-11 Security failure with SignonIncapable EPI test

```
C:\itsoctgv5\Java>java itso.cics.epi.SignonCapable tcp:\\volga scscpja4 cicsrs5
PASSWORD
Gateway URL: tcp://volga/
Region: SC62PJA4
Userid: CICSRS5
Password: PASSWORD

Adding signon incapable terminal
com.ibm.ctg.epi.EPIRequestException: CCL6799I: Return code EPI_ERR_SECURITY
from EPIRequest.addExTerminal call.
   at com.ibm.ctg.epi.Terminal.flowConnect(Terminal.java:461)
   at com.ibm.ctg.epi.Terminal.connect(Terminal.java:420)
   at itso.cics.epi.SignonIncapable.main(SignonIncapable.java:105)
```

Example 6-12 JES SYSLOG error for CTIN security violation

```
18.04.27 STC22530  ICH408I USER(CICSRS5 ) GROUP(SYS1    ) NAME(CICSRS5       )
   950               SCSCPJA4.CTIN CL(TCICSTRN)
   950               INSUFFICIENT ACCESS AUTHORITY
   950               ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE)
```

6.3 Problem determination

It is unlikely any implementation will be totally error free the first time you try things. In this section, we provide some hints and tips based on the experience we gained during this redbook project.

6.3.1 Tips and utilities

In this section, we provide some useful commands and utilities for debugging problems with your configuration, as well as some additional information about topics discussed in this chapter.

EXCI sign-on/sign-off message suppression

When running any application that connects to CICS through the EXCI (such as the CICS TG for z/OS), CICS will write one DFHSN1400/DFHSN1500 message pair to the CICS joblog. With the CICS TG for z/OS, this will occur each time an ECI request is flowed to the CICS TG. It may be desirable to suppress these sign-on messages, since so many can be produced. This could be done either by making the system equivalent (specifying a link user ID on the CICS SESSIONS definition) or by using the CICS user exit XMEOUT. An assembler sample of this program can be found in SDFHSAMP(DFH$SXP1).

Error messages

There are several pieces of software involved in securing CICS TG access to CICS. Because of this, error messages required for problem determination are seen in different locations. Below is a list of locations to look for error messages dealing with security problems:

CICS MSGUSR DD This DD, located in the active CICS region, can provide error messages having to do with VTAM connections to CICS. This also shows many security errors not shown anywhere else.

JES SYSLOG General security errors are displayed here. If you suspect a RACF error, search for the message ICH408.

Gateway daemon trace This provides an indication of the errors the Gateway daemon itself encounters.

CICS TG JNI logs These log files are automatically created in the $HOME/ibm/ctg directory and contain a log of all the EXCI errors received by the CTG JNI module.

CICS TG JNI trace This provides tracing of the EXCI calls made by the CICS TG JNI module libCTGJNI.so, and can be very

	useful in problem determination. This is a lower level of trace than the Gateway daemon trace. The logs are named after the pid number of the Gateway daemon process and so there is a unique log for each run of the CICS TG.
Java client trace	Turning on the Java client trace (in our case in the application EciB1) allows you to see the calls made to the CICS TG as well as the COMMAREA data.

Also helpful in problem diagnosis is the **`echo`** command. Including the **`echo`** command in the ctgstart script can be useful to help display the contents of variables. For example `the following statement` will display the setting of AUTH_USERID_PASSWORD:

```
echo AUTH_USERID_PASSWORD=$AUTH_USERID_PASSWORD
```

User not defined to RACF

Figure 6-13 is taken from the CICS MSGUSR DD. This shows user CICSRSX trying to access the system via IRP. CICSRSX is not defined to RACF.

Example 6-13 ACF access denied, user ID not defined

```
DFHSN1604 06/03/2002 22:55:33 SCSCPJA4 Attach header signon at terminal C31 by
user CICSRSX has failed. SAF codes are (X'00000004',X'00000000'). ESM codes are
(X'00000004',X'00000000').
DFHAC2047 06/03/2002 22:55:33 SCSCPJA4 While performing an attach for node
SCSCTG4 a security violation was detected.
```

In looking up DFHSH1604, we are informed SAF/ESM codes may be found in *External Security Interface (RACROUTE) Macro Reference for MVS*, GC23-3733. This manual tells us the error code for a SAF rc=4 and RACF rc=4 is a `user profile is not defined to RACF` error. Of course, you could have determined this by looking in the SYSLOG.

ECI return codes

The return codes for ECI calls, usually a negative number, can be found in the file header file cics_eci.h. This is not supplied on z/OS but can be found in the directory \ctg\include if using the CICS TG on Windows, AIX, Solaris, or OS/2. A summary of some common security errors we encountered is given below. For further details on errors with the CICS TG for z/OS, refer also to 7.4.1, “Tips and utilities” on page 164.

ECI_ERR_INVALID_DATA_LENGTH -1

This is a general error that notes that the CICS TG is unable to contact the CICS region. When this happens, a JNI trace (detailed in “EXCI connections to CICS” chapter) is often helpful.

The CICS TG JNI trace provides a UNIX errno return code. This is a standard UNIX style return code and on z/OS they are defined in the header file /usr/include/errno.h. In the following examples we show the JNI trace entry the errno.h entry and the actual reason we found for each error.

- **Errno 121**

  JNI trace entry `CCL6808I: CcicsECI: Authorize userid/password with RACF. Return code = -1, errno = 121.`

  errno.h #define EINVAL 121 /* Invalid argument

  Reason SURROGATE=YES but user ID and password null

- **Errno 139**

  JNI trace entry `CCL6808I: CcicsECI: Authorise userid/password with RACF. Return code = -1, errno = 139.`

  errno.h #define EPERM 139 /* Operation not permitted */

  Reason Security failure due to improper program control permissions

- **Errno 157**

  JNI trace entry `CCL6808I: CcicsECI: Authorize userid/password with RACF. Return code = -1, errno = 157.`

  errno.h #define EMVSERR 157 /* A MVS internal error has occurred */

  Reason SURROGATE=YES and JAVA_PROPAGATE=YES

- **Errno 163**

  JNI trace entry `CCL6808I: CcicsECI: Authorize userid/password with RACF. Return code = -1, errno = 163.`

  errno.h #define EMVSSAFEXTRERR 163 /* SAF/RACF extract error */

  Reason SURROGATE=YES and unauthorized RACF ID used

> **Tip:** The errno diagnostic information should only be used if the return code is not equal to 0. In the errors above, the return code is -1.

ECI_ERR_NO_CICS -3

This error can occur because:

- Inter-region communication (IRC) has not been started in the CICS region
- The EXCI CONNECTION definition is not installed in the CICS region
- The NETNAME in your EXCI connection does not match the value of the DFHJVPIPE variable referred to on the STDENV DD card in the CICS TG startup JCL.

ECI_ERR_SYSTEM_ERROR -9

This error generally implies that there is a problem with the EXCI communication mechanism. In our tests, we found this could be caused by two quite different situations:

- **Security attach failure for the mirror transaction**

  If you are using security and the user ID flowed in the ECI request does not have READ access to the TCICSTRN class profile controlling access to the specified mirror transaction, then you will receive this error. If this is the case, you will also receive the error message ICH408I in the CICS job log. For details on configuring the required profiles, refer to “Access to the mirror transaction, CSMI” on page 109.

- **MRO bind security failure**

  Access to DFHIRP is controlled by the DFHAPPL.<DFHJVPIPE> and DFHAPPL.<APPLID> profiles in the RACF FACILITY class.

  If you do not permit your CICS TG for z/OS address space user ID to log on to IRP, you will receive the error seen in Example 6-14. This was captured from a JNI trace. Details on getting this trace can be found in “JNI trace” on page 176.

Example 6-14 JNI trace of an unauthorized link

```
EXCI function error.  Function Call = 6, Response = 12, Reason = 414, Subreason
field-1 = 0, subreason field-2 = 0, Cics_Rc = -9
Error message from CICS: DFHAC2033 .
```

When you look up the response code 12, reason code 414, you see this translates to the message - IRP_ABORT_RECEIVED. You can find this message in the *CICS External Interfaces Guide*, SC34-6006.

ECI_ERR_SECURITY_ERROR -27

This error can be caused by several different situations, including the following:

- Surrogate security checking has failed. Check the MVS system console for RACF errors. For details on configuring surrogate security, refer to "Surrogate user security" on page 101.
- The CICS TG failed to authenticate the user ID and password specified in the ECI call. If you do not wish to authenticate this user ID and password within the CICS TG, ensure the variable AUTH_USERID_PASSWORD is not set in any of the following locations:
  - ctgenvvar
  - STDENV input member
  - The .profile of the CICS TG started task
  - /etc/profile
  - ctgstart script
- Program control should be enabled for SDFHEXCI. If this data set has an alias and you secure on the alias, you will see this error.

Part 3

Gateway daemon scenarios

In this section, we give details on how we configured the Gateway Transaction Gateway daemon on Windows 2000, OS/390, and Linux, and how we configured TCP/IP or Secure Sockets Layer (SSL) connections to the Gateway daemon from a remote Java client.

7

TCP connections to the Gateway daemon on z/OS

In this chapter, we describe how we installed and configured the CICS Transaction Gateway (CICS TG) on z/OS, and how we connected to it using the CICS TG TCP protocol, from a Java client application running on UNIX System Services and Windows 2000.

The CICS TG TCP protocol is a TCP/IP socket-based network protocol. It can be used by any remote Java application to communicate with the Gateway daemon. A UNIX daemon runs under UNIX System Services like a started task in MVS. In fact, the CICS TG daemon is started under an MVS started task in our implementation.

7.1 Preparation

The following sections detail the software levels we used in our sample configuration and instructions on how we installed the CICS TG for z/OS.

The communication protocol from the CICS TG for z/OS to the z/OS CICS region is External CICS Interface (EXCI). Because EXCI is complex enough to warrant its own chapter, we suggest you become familiar with Chapter 4, "EXCI connections to CICS" on page 65.

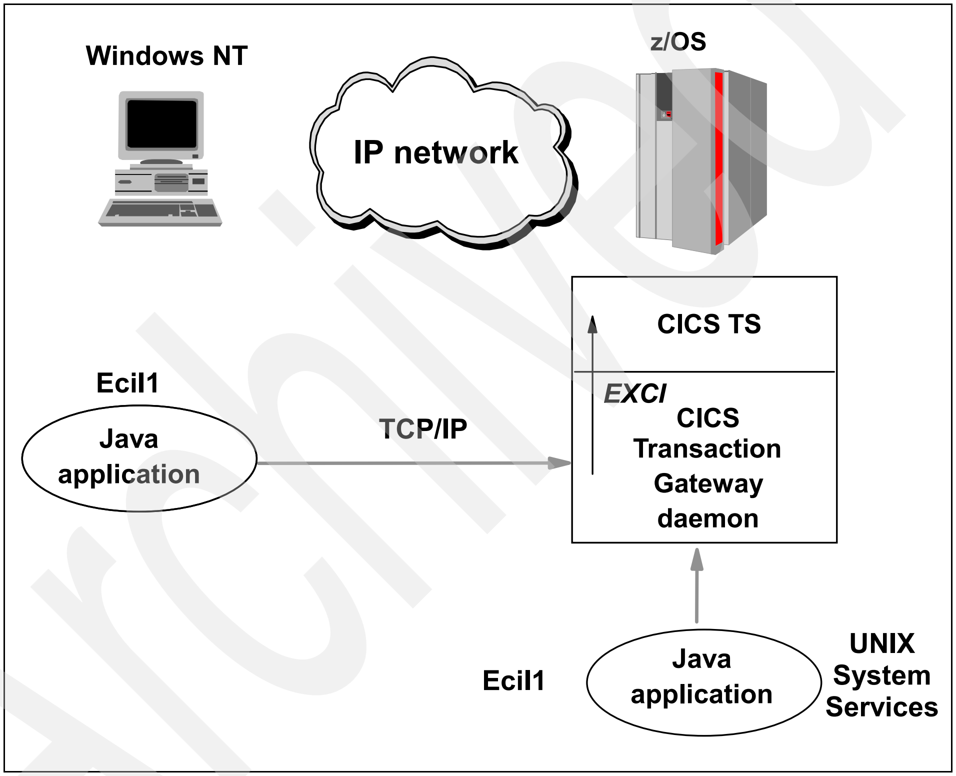


Figure 7-1 Software components: CICS TG for z/OS

The CICS TG for z/OS runs in the z/OS UNIX System Services environment. As such, you will need a mix of traditional MVS skills and UNIX skills in order to install and to configure it. In this chapter, we introduce the CICS TG and the UNIX System Services environment from the point of view of a CICS systems programmer with MVS skills who wishes to administer the CICS TG for z/OS.

7.1.1 Software checklist

We used the levels of software described in Table 7-1.

Table 7-1 Software checklist: CICS TG z/OS

Client workstation	z/OS
Windows 2000 Service Pack 2	z/OS V1.2
CICS Transaction Gateway for Windows V5.0.0	CICS Transaction Gateway for z/OS V5.0.0
IBM Java Development Kit V1.3.0	IBM Java Development Kit V1.3.1
	CICS Transaction Server V2.2

7.1.2 Definitions checklist

The definitions we used in this scenario are summarized in Table 7-2. Before you configure the products, we recommend that you acquire definitions for each parameter listed.

Table 7-2 Definitions checklist: CICS TG z/OS

CICS TG for z/OS	CICS Transaction Server	Example
hostname		wtsc66oe.itso.ibm.com
port		2006
Job name		SCSCTG5
	APPLID	SCSCPJA1

We had two TCP/IP stacks on our z/OS system: one for UNIX System Services and one for MVS. Most of our work was done with the UNIX System Services stack, primarily for FTP and the CICS TG for z/OS TCP/IP stack.

7.1.3 CICS configuration

While a great deal of our configuration has to do with setting up the CICS TG for z/OS, there are also CICS tasks that need to be performed.

Set up an EXCI connection

The CICS TG for z/OS uses the External CICS Interface (EXCI) protocol to communicate with CICS. EXCI is detailed in Chapter 4, "EXCI connections to CICS" on page 65 and shows how to set up the connections to your CICS region.

Compile and install the sample programs

We compiled the sample CICS COBOL programs, EC01 and EC02, and installed them in our CICS regions. These programs are provided with the CICS TG in the /usr/lpp/ctg500/ctg/samples/server directory. We copied them to our source library, compiled them, and put the load module in our CICS load library.

Edit and assemble DFHCNV

As we will be running our Java applications on Windows or UNIX System Services (both of which are ASCII environments), we will need to convert the EBCDIC CICS output to ASCII. To do this, we created a DFHCNV entry for EC01 and EC02 in our CICS DFHCNV table. This meant that CICS would convert the input for these programs from ASCII to EBCDIC, and the output from EBCDIC to ASCII. For further details on using DFHCNV, refer to Appendix A, "DFHCNV and CICS data conversion" on page 329.

7.1.4 Installation of the CICS TG

The following section details how we installed the CICS TG on our z/OS system.

Transferring the CICS TG files to z/OS

The CICS TG for z/OS V5.0.0 is supplied on a CD-ROM in the directory x:\GATEWAYS\OS390\ctg-500m.tar.Z. We transferred it from Windows to our z/OS system in binary mode using the File Transfer Protocol (FTP).

We used the UNIX System Services FTP started task to upload the file directly into the /tmp HFS directory on our z/OS system. Example 7-1 on page 137 shows the output of our FTP session. We used binary mode for the transfer, since we did not want FTP to perform data conversion on the tar archive.

Example 7-1 Using FTP to send the z/OS gateway to the /tmp= directory

```
C:\>ftp wtsc66oe.itso.ibm.com
Connected to wtsc66oe.itso.ibm.com.
220-FTPDOE1 IBM FTP CS V1R2 at wtsc66oe.itso.ibm.com, 18:50:46 on 2002-06-13.
220 Connection will not timeout.
User (wtsc66oe.itso.ibm.com:(none)): cicsrs2
331 Send password please.
Password:
230 CICSRS2 is logged on.  Working directory is "/u/cicsrs2".
ftp> cd /tmp
250 HFS directory /tmp is the current working directory
ftp> lcd d:\gateways\os390
Local directory now D:\gateways\OS390.
ftp> bin
200 Representation type is Image
ftp> put ctg-500m.tar.Z
200 Port request OK.
125 Storing data set /tmp/ctg-500m.tar.Z
250 Transfer completed successfully.
ftp: 21256941 bytes sent in 60.14Seconds 353.48Kbytes/sec.
ftp> quit
221 Quit command received. Goodbye.
```

Tip: If you only have an MVS FTP server, you can FTP the CICS TG tar file to an MVS data set name using a binary transfer. The record format of the MVS data set is not important, but it should be allocated with at least 30 cylinders.

Once the CICS TG compressed file is in an MVS data set, you can move it over to UNIX System Services using one of the following methods:

Using ISHELL	Use **File -> New** and **File -> Replace From**
Using UNIX System Services	Use the command **`tso oget`**
Using TSO	Use the command **`TSO OPUT`**
Using batch TSO	Use OPUT

Again, regardless of which method you use, you should specify binary format.

Once the CICS TG install archive was in the UNIX System Services /tmp directory, we created an environment where we could install the CICS TG for z/OS. Many of these tasks can also be performed by running the JCL found in /usr/lpp/ctg500/ctg/samples/jcl. The file /usr/lpp/ctg500/ctg/samples/samples.txt explains the members in this directory.

To create our environment, we performed the following steps.

HFS data set creation

We created a new HFS and mounted it at the directory /usr/lpp/ctg500, where we installed the CICS TG code.

To set up an HFS, you can allocate it using either ISPF option 3.2, an IEFBR14 batch job, or through the ISHELL menus. You should make sure that the storage class with which your file is allocated has the GUARANTEED SPACE attribute. For our /usr/lpp/ctg500 directory we created a data set with a data set name of CTGUSER.CTG500.HFS. Figure 7-2 shows how we set up our HFS using TSO option 3.2.

```
  Menu  RefList  Utilities  Help
------------------------------------------------------------------------------
                           Allocate New Data Set
Command ===> _________________________________________________________________

Data Set Name  . . . : CTGUSER.CTG500.HFS

Management class . . . ________       (Blank for default management class)
Storage class  . . . . ________       (Blank for default storage class)
 Volume serial . . . . tsms01         (Blank for system default volume) **
 Device type . . . . . ________       (Generic unit or device address) **
Data class . . . . . . ________       (Blank for default data class)
 Space units . . . . . CYLINDER       (BLKS, TRKS, CYLS, KB, MB, BYTES
                                       or RECORDS)
 Average record unit   _              (M, K, or U)
 Primary quantity  . . 50             (In above units)
 Secondary quantity    50             (In above units)
 Directory blocks  . . ______         (Zero for sequential data set) *
 Record format . . . . U
 Record length . . . . 0
 Block size  . . . . . 0
 Data set name type  : HFS            (LIBRARY, HFS, PDS, or blank)  *
                                      (YY/MM/DD, YYYY/MM/DD
 Expiration date . . . __________      YY.DDD, YYYY.DDD in Julian form
Enter "/" to select option             DDDD for retention period in days
_  Allocate Multiple Volumes           or blank)

( * Specifying LIBRARY may override zero directory block)

( ** Only one of these fields may be specified)
```

Figure 7-2   ISPF display of the CTGUSER.CTG500.HFS data set

Once the HFS data set has been allocated, you must mount it at an HFS directory, known as a *mount point*, with a TSO `MOUNT` command or by using Option 3 from the ISHELL File_systems menu. The mount point should be an empty directory; otherwise its contents will be hidden (but not deleted).

We created our directory in OMVS using superuser authority as shown in Example 7-2 on page 139. First we changed to the /usr/lpp directory and issued the **su** command to switch users to the superuser. Next we made our ctg500 directory and listed it with the **ls** command.

Example 7-2 Creating a mount point directory

```
$ SC66¨ /u/cicsrs2: cd /usr/lpp
$ SC66¨ /usr/lpp: su
$ SC66¨ /usr/lpp: mkdir ctg500
$ SC66¨ /usr/lpp: ls -l ctg500
drwxrwxrwx   2 AAAAAAA  OMVSGRP     8192 Jun 20 10:27 ctg500/
```

After creating the `/usr/lpp/ctg500` directory, we mounted our HFS onto this using the ISHELL File_systems menu (Figure 7-3).

```
                          Mount a File System

 Mount point:
                                                      More:     +
    /usr/lpp/ctg500


 File system name . . ctguser.ctg500.hfs
 File system type . . hfs
 New owner  . . . . .
 Owning system  . . .           Character Set ID . .

 Select additional mount options:
 _  Read-only file system
 _  Ignore SETUID and SETGID
 _  Bypass security
 _  Do not automove file system
 _  Text conversion enabled
 Mount parameter:
```

Figure 7-3 ISHELL, Mount a File System screen

If you want your mounted HFS file systems to be available when your z/OS system is initial program loaded, you will need to add `MOUNT` statements to your BPXPRMxx parameters in SYS1.PARMLIB. For example these are the parameters we added to our z/OS system for our /usr/lpp/ctg500 HFS:

```
MOUNT  FILESYSTEM('CTGUSER.CTG500.HFS')  TYPE(HFS)
MOUNTPOINT('usr/lpp/ctg500')  MODE(RDWR)
```

Unpacking the CICS TG filesets

Once we had created the /usr/lpp/ctg500 HFS, we uncompressed the CICS TG tar archive. From OMVS, we uncompressed the tar archive using the UNIX **uncompress** command as follows:

```
cd /tmp
uncompress -v /tmp/ctg-500m.tar.Z
```

This created an uncompressed tar archive file, ctg-500m.tar (without the ".Z" suffix), in the /tmp directory. We unpacked this archive into the new /usr/lpp/ctg500 directory using the tar command:

```
cd /usr/lpp/ctg500
tar -xopfv /tmp/ctg-500m.tar
```

This created the appropriate subdirectories and installed the CICS TG into the /usr/lpp/ctg500 directory.

For reference, the options on the tar command are as follows:

o: Does not restore the files with the original owner and group IDs, but instead uses the owner and group of the current logged-in user. This is important, since the original owner of the files will not be known on your z/OS UNIX System Services.

p: Restores the files with the high-order file permission bits (set-user-ID, set-group-ID, and sticky bit).

f: Specifies file name of the input tar archive file.

v: Specifies verbose messages.

Running the CICS TG install script

We ran the next section in TSO OMVS, which is probably more familiar to CICS systems programmers than a telnet session to UNIX System Services.

In the /user/lpp/ctg500 directory, the `ls -l` command showed the following files:

```
drwxr-xr-x   9 AAAAAAA  SYS1       8192 Jun 21 00:04 ctg/
-rwxr-xr-x   1 AAAAAAA  SYS1         66 Jun 21 00:05 ctg_install
-rwxr-xr-x   1 AAAAAAA  SYS1         20 Jun 21 00:07 ctginstall
-rw-r--r--   1 AAAAAAA  SYS1       1880 Jun 21 00:05 ctginstall.readme
-rwxr-xr-x   1 AAAAAAA  SYS1         28 Jun 21 00:07 ctginstall_IBM-930
-rwxr-xr-x   1 AAAAAAA  SYS1         28 Jun 21 00:07 ctginstall_IBM-935
-rwxr-xr-x   1 AAAAAAA  SYS1         28 Jun 21 00:07 ctginstall_IBM-937
-rwxr-xr-x   1 AAAAAAA  SYS1         28 Jun 21 00:07 ctginstall_IBM-939
```

You will need to run the ctginstall script to install the CICS TG. If you have problems reading the license agreement, you can enter **`ctginstall 64`**. You are presented with a page of licensing agreements. Enter 3 to accept the agreement and complete the installation. A partial listing of the installation process is shown in Example 7-3 on page 141.

> **Tip:** When running ctginstall under TSO OMVS, at some point you will probably see `INPUT` in the lower right-hand corner of your screen. This means OMVS has put your terminal to sleep. Press PF10 to refresh the screen.

Example 7-3 CICS TG install script

```
CICSRS2 @ SC66:/usr/lpp/ctg500>ctginstall
To install CICS Transaction Gateway you must accept the following license
agreement. If the license file does not display correctly, try restarting
the installation using the command:
'/usr/lpp/ctg500/ctg/bin/ctgsetup 64'.
Enter 1 to view the license, or 2 to quit.
1
>>> Beginning of the License File >>>
International Program License Agreement
Part 1 - General Terms
PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THE PROGRAM. IBM WILL LICENSE
THE PROGRAM TO YOU ONLY IF YOU FIRST ACCEPT THE TERMS OF THIS AGREEMENT. BY
USING THE PROGRAM YOU AGREE TO THESE TERMS. IF YOU DO NOT AGREE TO THE TERMS OF
THIS AGREEMENT, PROMPTLY RETURN THE UNUSED PROGRAM TO THE PARTY (EITHER IBM OR
ITS RESELLER) FROM WHOM YOU ACQUIRED IT TO RECEIVE A REFUND OF THE AMOUNT YOU
PAID.
______________________________________________________________________________
Enter 1 to page down, 2 to page up, 3 to accept or 4 to decline.
===>  3
You have accepted the license agreement.
Wait while the installation continues.
The installation is complete.
License files can be viewed using the command:
'/usr/lpp/ctg500/ctg/bin/ctgbrowse'
To view the PDF documentation, you should obtain a copy of Adobe Acrobat Reader
from www.adobe.com. Now create 'CTG.INI' and 'ctgenvvar' by copying
'CTGSAMP.INI' and 'ctgenvvarsamp' and editing them to configure the CICS
Transaction Gateway.
 ===>                                                                  INPUT
```

The result of our CICS TG install was the directory structure shown in Figure 7-4 on page 142.

.

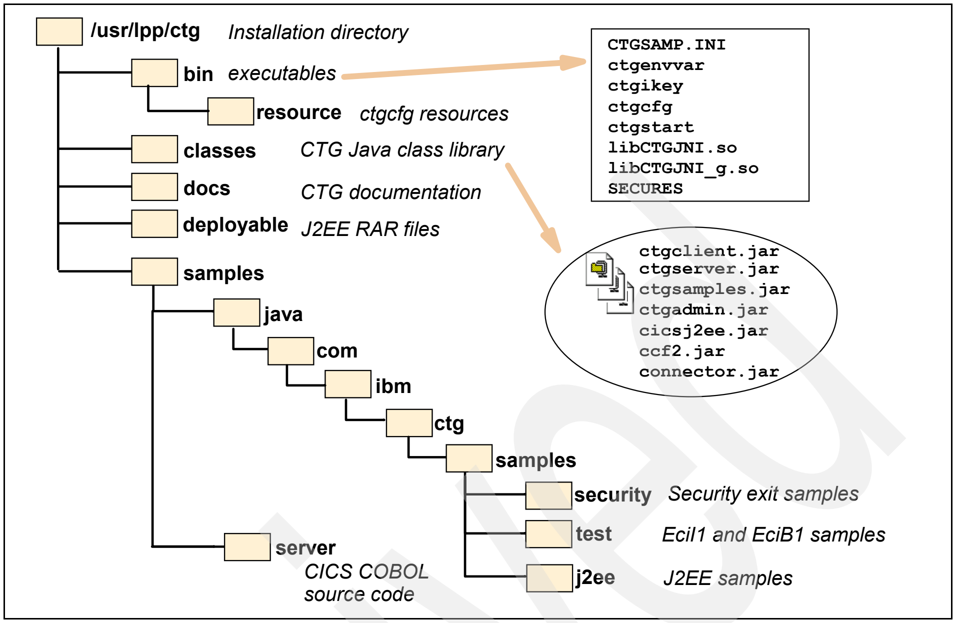


Figure 7-4 CICS TG z/OS directory structure

The directory /usr/lpp/ctg500/ctg/bin contains the following files, all of which can be browsed using TSO `OBROWSE` or `ISHELL`. The other files in the directory are executables and cannot be viewed.

CTGSAMP.INI Sample CTG.INI configuration file.

ctgenvvar Environment variables (created during a later step from ctgenvvarsamp)

ctgcfg Shell script to start CICS TG X-Windows graphical Configuration Tool.

libCTGJNI.so Shared library called by the CICS TG (using the JNI) to invoke the EXCI.

ctgstart Shell script to start the Gateway daemon.

The /usr/lpp/ctg500/ctg/classes directory contains the following JAR files:

ctgclient.jar	Java class library
ctgserver.jar	Classes for use by Gateway daemon
ctgsamples.jar	Samples
ctgadmin.jar	Trace admin client
cicsj2ee.jar, ccf2.jar, connector.jar	J2EE classes

> **Tip:** The contents of a JAR or zip file can be listed with the UNIX System Services command:
>
> ```
> jar -tvf <file>
> ```

Basic security considerations

Before we started our CICS TG started task, we had to set up basic RACF security. These are the required RACF tasks related to enable the CICS TG:

- Enable RACF program control checking
- Define the SDFHEXCI library as program controlled to RACF
- Allow the CICS TG user ID read access to program-controlled libraries
- Remove address space sharing from ctgstart using **`extattr -s`**
- Create a user ID for the CICS TG and assign it to our started task name
- Mark UNIX System Services programs as program controlled with the **`extattr +p`** command

These tasks will allow the most basic RACF security permissions. For a more secure environment, see Chapter 6, “CICS TG security scenarios” on page 99.

Enable RACF program control checking

To use RACF security functions within the CICS TG, all data sets, or HFS files, the CICS TG uses must be marked as program controlled. In the UNIX System Services context, program control allows RACF to secure UNIX System Services executables as if they were MVS programs.

Before marking data sets and UNIX System Services executables as program control, we have to turn on program control in RACF. We activated program control by issuing these commands from a user ID with RACF SPECIAL authority:

```
SETROPTS CLASSACT(PROGRAM)
RDEFINE PROGRAM *UACC(READ)
SETROPTS WHEN(PROGRAM)
```

Define the SDFHEXCI library as program controlled to RACF

Since the CICS TG runs from UNIX System Services program controlled executables, any MVS library the CICS TG accesses must also be program controlled. This includes the SDFHEXCI library which can be marked as program controlled with the following RACF command:

```
RALTER PROGRAM * ADDMEM(‘CICSTS22.CICS.SDFHEXCI’//NOPADCHK)
SETROPTS WHEN(PROGRAM) REFRESH
```

We encountered a problem with program control on this library. This problem may have been specific to our site, but bears mentioning. The SDFHEXCI library specified in the CICS TG started task and in the CICS region was an alias. Security failed when the CICS TG accessed the actual data set name. Defining the actual data set name solved our problem.

We also had a problem when running with security enabled on the CICS TG. The EciI1 example would give us *dirty address space* errors when attempting to access module DFHXCSVC in the SDHFLINK library. Marking CICSTS22.CICS.SDFHLINK as program controlled solved this problem.

Allow read access to program controlled libraries

If your z/OS system has defined the BPX.SERVER FACILITY class profile within RACF, then the user ID under which the Gateway daemon runs must be permitted to this profile. The following example shows the `PERMIT` command assuming the user ID of the server is CTGUSER:

```
PERMIT BPX.SERVER CLASS(FACILITY) ID(CTGUSER) ACCESS(READ)
PERMIT BPX.FILEATTR.PROGCTL CLASS(FACILITY) ID(CTGUSER) ACCESS(READ)
```

If the BPX.SERVER FACILITY class is not defined, the CICS TG user ID must be defined with a UID of 0 (that is, be a root user).

Remove address space sharing from ctgstart

We specified that the ctgstart script should not share its address space with any other processes. This is to ensure that the calling address space is not contaminated by a non-program-controlled load module. To force the JVM to use its own non-shareable address, we used the following UNIX System Services command:

```
extattr -s /usr/lpp/ctg500/bin/ctgstart
```

This command should be performed from the owner ID or superuser. Example 7-4 on page 145 shows the result of this command. The ctgstart module does not have an “s” in the second column, which is the desired effect of the above command.

CICS TG started task user ID

We ran our CICS TG as a started task, using the user ID CTGUSER. The user ID under which the CICS TG started task runs should:

- Have an OMVS segment defined
- Be in a group that has an OMVS segment
- Be defined without a password
- Have READ access to the RACF profile that protects the TCPIP.STANDARD.TCPXLBIN data set. This data set contains translate tables for translating from ASCII to EBCDIC and from EBCDIC to ASCII.

There are two methods of defining a default user ID to a started task:

- Use the RACF exit - ICHRIN03
- Use the STARTED class in RACF. The following example shows the user ID CTGUSER, in the CICS group, being assigned as the user ID for all SCSCTG* tasks:

```
RDEF STARTED SCSCTG*.* STDATA(USER(CTGUSER) GROUP(CICS) PRIVILEGED(NO)
TRUSTED(NO))
```

More information can be found in the *RACF Systems Programmer's Guide*, SC28-1913.

Program control

To use RACF security functions within the CICS TG, all data sets, or HFS files, the CICS TG uses must be marked as program controlled. In the UNIX System Services context, program control allows RACF to secure UNIX System Services executables as if they were MVS programs. If you have not turned on program control for the required data sets, you will probably get 02AF (address bit dirty) abends.

The UNIX System Services command **`extattr +p`** is then used to mark UNIX System Services files as program controlled. You need to be the superuser to modify the file permissions, so it is best to issue the **`extattr`** command as superuser.

These commands mark the CICS TG files as program controlled:

```
extattr +p /usr/lpp/ctg500/ctg/bin/lib*.so
extattr +p /usr/lpp/ctg500/ctg/bin/SECURES
```

The following command marks Java files that the CICS TG requires to be program controlled:

```
extattr +p /usr/lpp/java/IBM/J1.3/bin/*
```

To verify the program control commands worked, you can issue the **`ls -E`** command from a UNIX System Services shell. Example 7-4 is an example of the output of this command. The `-p` in the second column shows that program control is set.

Example 7-4 CICS TG program controlled files

```
CICSRS2 @ SC66:/usr/lpp/ctg500/ctg/bin>ls -E
total 20072
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       84990 Jun 21 00:04 CTGLOG.HLP
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       57410 Jun 25 12:18 CTGMSG.HLP
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       12662 Jun 21 00:04 CTGSAMP.INI
-rwxr-xr-x  -ps-  1 CTGUSER  SYS1      139264 Jun 21 00:04 SECURES
```

```
-rwxr-xr-x  a-s-  1 CTGUSER  SYS1       12288 Jun 21 00:04 ctgarm
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       31097 Jun 21 00:04 ctgbrowse
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       34026 Jun 21 00:04 ctgcfg
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       27533 Jun 21 00:04 ctgdoc
-rwxr-xr-x  --s-  1 CTGUSER  SYS1         923 Jun 25 14:26 ctgenvvar
-rwxr-xr-x  --s-  1 CTGUSER  SYS1        5958 Jun 21 00:04 ctgenvvarsamp
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       36366 Jun 21 00:04 ctgikey
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       48936 Jun 21 00:04 ctgmsgs
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       40069 Jun 21 00:06 ctgsetup
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       40069 Jun 21 00:06 ctgsetup_IBM-930
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       40069 Jun 21 00:06 ctgsetup_IBM-935
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       40069 Jun 21 00:06 ctgsetup_IBM-937
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       40069 Jun 21 00:06 ctgsetup_IBM-939
-rwxr-xr-x  ----  1 CTGUSER  SYS1       36854 Jun 25 15:00 ctgstart
-rwxr-xr-x  --s-  1 CTGUSER  SYS1       31075 Jun 21 00:04 ctguninst
-rwxr-xr-x  -ps-  1 CTGUSER  SYS1      348160 Jun 21 00:04 libCTGJNI.so
-rwxr-xr-x  -ps-  1 CTGUSER  SYS1      348160 Jun 21 00:04 libCTGJNI_g.so
-rwxr-xr-x  -ps-  1 CTGUSER  SYS1      180224 Jun 21 00:04 libSSLSocketLib.so
-rwxr-xr-x  --s-  1 CTGUSER  SYS1        2349 Jun 21 00:04 oldctgcfg
-rwxr-xr-x  --s-  1 CTGUSER  SYS1        4088 Jun 21 00:04 oldctgikey
-rwxr-xr-x  --s-  1 CTGUSER  SYS1        6040 Jun 21 00:04 oldctgstart
-rwxr-xr-x  --s-  1 CTGUSER  SYS1        1929 Jun 21 00:04 readme.txt
drwxr-xr-x       11 CTGUSER  SYS1        8192 Jun 21 00:04 resource
```

7.2 Configuration

In this section, we explain how we configured the CICS TG daemon to be started as a started task and to listen for incoming ECI requests on a given TCP/IP port. We used the name SCSCTG5 for our CICS TG.

1. We created a HFS working directory for our CICS TG SCSCTG5 and its log files called /ctg/scsctg5/logs. To do this we used the same procedure as used for creating the /usr/lpp/ctg500 HFS outlined in “HFS data set creation” on page 138. This enabled us to separate our CICS TG log files from the product install files and thus prevented the possibility that the /usr/lpp/ctg500 HFS could become full due to trace output.
2. We copied the sample configuration file /usr/lpp/ctg500/ctg/CTGSAMP.INI to /ctg/scsctg5/CTG.INI.
3. We activated the CICS TG TCP protocol handler by uncommenting the lines in CTG.INI as shown in Example 7-5 on page 147. The continuation character in this file is a backslash.

Example 7-5 TCP protocol handler

```
SECTION GATEWAY
   closetimeout=10000
   ecigenericreplies=off
   initconnect=10
   initworker=10
   maxconnect=90
   maxworker=90
   noinput=off
   nonames=on
   notime=off
   workertimeout=10000

   protocol@admin.handler=com.ibm.ctg.server.RestrictedTCPHandler
  protocol@admin.parameters=port=2810;\
       sotimeout=1000;connecttimeout=2000;idletimeout=600000;\
       pingfrequency=60000;maxconn=5;
#       allowaddr=127.0.0.1;
  protocol@tcp.handler=com.ibm.ctg.server.TCPHandler
  protocol@tcp.parameters=connecttimeout=2000;idletimeout=600000;\
       pingfrequency=60000;port=2006;solinger=0;sotimeout=1000;
ENDSECTION
```

The TCP/IP port 2006 is the default port for the TCP/IP listener and, since no other Gateway daemons were running in the LPAR, we used this value.

We set up the administration client so that we can control the traces dynamically. There was no other administration client, so we kept the default port of 2810.

4. Next, we created the JCL to allow us to start the CICS TG as an MVS started task (Example 7-6).

Example 7-6 JCL for CICS TG started task

```
//SCSCTG5 EXEC PGM=BPXBATCH,REGION=0M,
//         PARM='SH /usr/lpp/ctg500/ctg/bin/ctgstart -noinput'
//STEPLIB  DD DSN=CICSTS22.CICS.SDFHEXCI,DISP=SHR
//STDIN    DD PATH='/dev/null',PATHOPTS=(ORDONLY)
//STDOUT   DD PATH='/ctg/scsctg5/logs/ctg.stdout',
//            PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
//            PATHMODE=(SIRWXU,SIRWXG,SIROTH)
//STDERR   DD PATH='/ctg/scsctg5/logs/ctg.stderr',
//            PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
//            PATHMODE=(SIRWXU,SIRWXG,SIROTH)
//STDENV   DD DSN=ESA.SYS1.PROCLIB(CTG5ENV),DISP=SHR
```

BPXBATCH is the MVS program that runs a UNIX System Services script as a batch job. The PARM field specifies that the shell (SH) is to execute the specified **`ctgstart`** command with the `-noinput` option.

We directed the STDOUT and STDERR files to a location different from the base CICS TG software. This allowed us to manage the logs in a different directory, and in this case, an entirely different HFS data set.

The STDENV DD points to a PDS member. You can administer the parameters from TSO, which is easier for CICS systems programmers. The concatenation sequence for ctgstart, ctgenvvar, and STDENV DD are noted in 7.2.2, “Defining CICS TG configuration parameters” on page 153.

PATHOPTS on the HFS file specification is similar to the JCL DISP parameter, and PATHMODE determines UNIX file permissions for the file.

7.2.1 Defining CICS TG environmental variables

Environment variables passed from BPXBATCH to the Gateway daemon can be set in one of the following three ways:

- Within the JCL using a partitioned data set (PDS) referenced by the STDENV DD card.
- Using the ctgenvvar HFS file.
- Within the shell environment of the started task user ID.

The UNIX search order for environment variables used by BPXBATCH is as follows:

1. Variables found in the /usr/lpp/ctg500/ctg/bin/ctgenvvar file take precedence.
2. Variables are next taken from the UNIX System Services shell environment (BPXBATCH) in which ctgstart runs. The order of precedence is as follows:
   a. Variables taken from the .profile of the started task user ID
   b. Variables taken from the default user profile /etc/profile
   c. Variables taken from STDENV member passed to BPXBATCH

This is summarized in Figure 7-5.

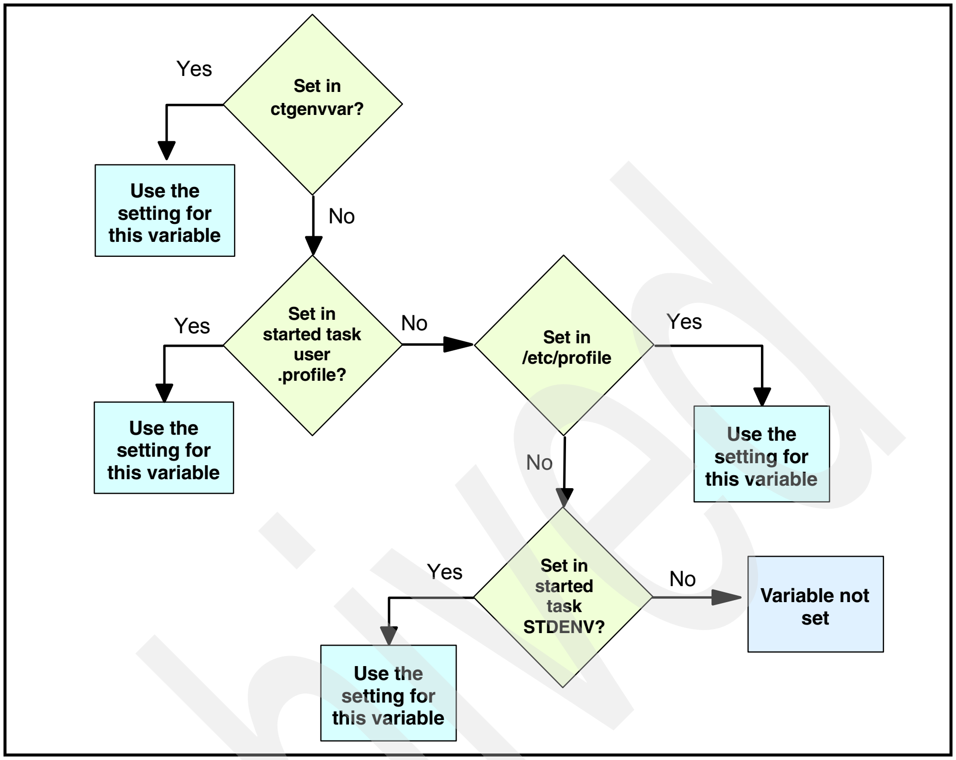


Figure 7-5 Concatenation sequence for CICS TG variables

We spent most of our time in TSO, and to ease maintenance we set most of our variables in the STDENV member defined in the CICS TG started task. To ensure these variables were not overridden, we also checked the variables were not set in the files: ctgenvvar, /etc/profile or /u/ctguser/.profile.

> **Note:** In versions of the CICS TG prior to V5.00, environment variables could also be set in the ctgstart script itself. This is not recommended and you should note that the code to allow this has now been removed from ctgstart.

The STDENV library member (Example 7-7) contained our environment variables to be passed to the ctgstart script.

Example 7-7 CICS TG STDENV member

```
DFHJVPIPE=SCSCTG5
DFHJVSYSTEM_00=SCSCPJA1 - CICSTS 2.2
DFHJVSYSTEM_01=SCSCPJA2 - CICSTS 2.2
EXCI_LOADLIB=CICSTS22.CICS.SDFHEXCI
```

```
CICSCLI=/ctg/scsctg5/CTG.INI
CTG_RRMNAME=CCL.CTG.IBM.UA
CTGSTART_HOME=/usr/lpp/ctg500/ctg/bin
```

The CICS TG V5 supplies a new and somewhat complex ctgenvvarsamp file as a sample of the ctgenvvar file used for controlling of environment variables. This new ctgenvvar will override variables set in other places, such as STDENV. To understand this new ctgenvvar file, you will need to be familiar with UNIX System Services and Korn shell scripting. If you are not, we suggest you use a simple version, such as the one we used (shown in Example 7-8), which does not override variables set in other places.

Example 7-8 ctgenvvar

```
CTG_CLASSPATH="${CTGSTART_HOME}/../classes/ctgclient.jar:${CTGSTART_HOME}/../cl
asses/ctgserver.jar:${CTGSTART_HOME}/../classes/cfwk.zip:${CTGSTART_HOME}/../cl
asses/ctgsamples.jar"
CTG_LIBPATH="${CTGSTART_HOME}"
export STEPLIB=${STEPLIB}:${EXCI_OPTIONS}:${EXCI_LOADLIB}
export CLASSPATH=${CTG_USER_CLASSPATH}:${CTG_CLASSPATH}:$CLASSPATH
export LIBPATH=${CTG_USER_LIBPATH}:${CTG_LIBPATH}:$LIBPATH
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${CTG_USER_LIBPATH}:${CTG_LIBPATH}
export PATH=.:/bin:/usr/lpp/java213/J1.3/bin
export JAVA_HOME=/usr/lpp/java213/J1.3
```

Following is a summary of the variables you are likely to have to set:

- **DFHJVPIPE**

  The value specified here should match the NETNAME in the CICS connection definition to be used for an EXCI connection using a specific pipe. The default connection name in the IBM-supplied group DFH$EXCI is EXCS, which has a NETNAME of BATCHCLI. We created our own connection definition, and used a NETNAME of SCSCTG5.

- **DFHJVSYSTEM**

  The syntax of this variable is DFHJVSYSTEM_*nn=aaaaaaaa literal*, where:

  – *nn* = 0 to 99
  – *aaaaaaaa* = the APPLID of the CICS region
  – *literal* = a description of the CICS region

  Note the setting of this variable is optional, and only affects the result of the `ECIRequest.listSystems()` method and does not affect which CICS regions you can connect to.

- **AUTH_USERID_PASSWORD**

  This operand determines whether the user ID and password sent on an ECI call should be verified by RACF before the EXCI call is made. The default is YES. We did not wish to flow user IDs on the ECI calls in our tests, and therefore left this line commented out in our ctgstart script, ctgenvvar & STDENV member.

> **Tip:** The way CICS TG checks AUTH_USERID_PASSWORD is unusual. While MVS checks to see the contents of a variable, the CICS TG ignores the contents of this variable and merely checks if it is null or not null. AUTH_USERID_PASSWORD can be YES, NO, or REDRUM and it is still considered set (not null).
>
> To unset a variable, comment it out in ctgenvvar and your STDENV member, and make sure it is not set in /etc/profile or the .profile of the user ID under which the CICS TG is running.

- **EXCI_LOADLIB**

  This is the name of the library containing the EXCI load modules, without its high-level qualifier. This will usually be SDFHEXCI, and so does not usually need modifying.

- **EXCI_OPTIONS**

  This is a user library that contains the EXCI options to be used if you have a tailored version of the EXCI options table (DFHXCOPT). For more details, refer to Chapter 6, “CICS TG security scenarios” on page 99.

- **HLQ**

  This is the high-level qualifier for the library specified in EXCI_LOADLIB. If you do not have your CICS installation image in the default location CICSTS22.CICS, then you will need to modify this statement.

- **CICSCLI**

  You can specify a different path to the configuration file by setting the location of CTG.INI. This variable is instrumental in running multiple CICS TG started tasks.

  You will note in our example that we point the CICS TG to a directory specific to the started task. This allows each CICS TG started task to have its own CTG.INI.

- **CTG_CLASSPATH**

  This controls the classpath used by the Gateway daemon. It is best set in ctgenvvar, and needs modifying only if you need to make a server compression class available to a protocol handler. This will be the case if you use the Ecil1 transactional ECI sample, which uses classes supplied in ctgsamples.jar. To enable use of Ecil1, we set the CTG_CLASSPATH in ctgenvvar as follows:

```
CTG_CLASSPATH="${CTGSTART_HOME}/../classes/ctgclient.jar:${CTGSTART_HOME}
/../classes/ctgserver.jar:${CTGSTART_HOME}/../classes/cfwk.zip:
${CTGSTART_HOME}/../classes/ctgsamples.jar"
```

- **CTG_RRMNAME**

  This is the name that the CICS TG registers with Resource Recovery Services (RRS) for transactional EXCI requests to CICS. You will only need to modify this if using transactional EXCI requests. The default RRMNAME is CCL.CTG.IBM.UA. If you choose not to use the default name, you must obey the naming rules for RRS groups.

  The name can consist of the following printable characters:

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

  RRM name restrictions include:

  – The name cannot start with a blank or contain embedded blanks.
  – Lowercase characters are converted to uppercase characters.
  – To avoid naming conflicts, use A-C or G-I as the first character.
  – The length of CTG_RRMNAME must not exceed 32 characters and must end with the characters IBM.UA

Transactional EXCI: Transactional EXCI is an application programming interface (API) that allows the CICS server unit of work to be continued over multiple EXCI calls, until the EXCI client decides to commit or back out the unit of work. This means that multiple ECI requests from the CICS TG for z/OS in Extend_Mode can be part of the same logical unit of work, and can be coordinated using a two-phase commit mechanism from the CICS TG for z/OS to the CICS server.

The EXCI requests can only be transactional if the CICS TG started task using the EXCI and the CICS region execute in the same z/OS image.

7.2.2 Defining CICS TG configuration parameters

There are several CICS TG configuration parameters that can be modified. We modified these by editing the CTG.INI file. You can use the X-Windows **ctgcfg** tool if you have X-Windows.

> **Note:** The CICS TG Configuration Tool utility **ctgcfg** is supplied with the CICS TG for z/OS and can be used on z/OS. However, you will need to be familiar with X-Windows and have configured an X client to be able to use this utility. If you are not familiar with UNIX and X-Windows, you could simply edit the CTG.INI file in the /usr/lpp/ctg500/ctg/bin directory.
>
> Another method is to enter **ctgcfg -PLAT ZOS** on a Windows workstation that has the CICS TG installed. This uses the CICS TG for Windows configuration utility to create z/OS configuration files. Once you've saved the CTG.INI file, you simply FTP it to the location in your CICSCLI variable.

Changes to any of these parameters requires a stop and restart of the CICS TG before they will take effect. The following is a list of the most important configuration parameters, along with the values we used:

- **initconnect**=10

  This is the initial number of connection manager threads allocated at startup. Set this to the "normal" number of clients you expect to support.

- **initworker**=10

  This is the initial number of worker threads allocated at startup. Set this to the initial number of connection manager threads.

- **maxconnect**=100

  This is the maximum number of connection manager threads. This limits the maximum number of remote Java clients that can connect to the CICS TG through one of the protocol handlers. Set this to the maximum number of clients you need to support, but also make sure it is greater than or equal to maxworker.

- **maxworker**=100

  This is the maximum number of worker threads, which limits the maximum number of parallel ECI request the CICS TG can process. It should be:

  – Less than or equal to maxconnect

  – Less than or equal to the RECEIVECOUNT set in the CICS sessions definition

  – Less than or equal to 100, since this is the maximum number of EXCI pipes that any z/OS address space can allocate

- **noinput=**no

  This means the ctgstart script is able to read input if it is started from the UNIX System Services shell. If ctgstart is run as a started task, this parm has no effect.

- **nonames**=on

  This controls whether client host names are resolved via the DNS server. This should be set to `on` to prevent delays.

- **notime**=off

  This shows the timestamps in the message log. This should be set to `off`, since timestamps are very useful in problem determination.

- **tfile**=`/ctg/scsctg5/logs/ctg.trc`

  This is the location of the trace file. We set it to our CICS TG logging directory we created in 7.2, "Configuration" on page 146.

- **workertimeout**=1000

  This is the time in milliseconds a connection manager thread waits to obtain a thread from the worker thread pool. This timeout prevents remote Java client from hanging if all worker threads are in use.

7.2.3 EXCI pipe usage

The EXCI is used by the CICS TG for z/OS to send requests to the CICS region specified in the ECI request. To make an ECI call, the CICS TG for z/OS uses the EXCI CALL interface. The EXCI CALL interface provides a low-level API for calling a CICS program using a COMMAREA, and consists of the following six commands:

1. Initialize_User
2. Allocate_Pipe
3. Open_Pipe
4. DPL_Request
5. Close_Pipe
6. Deallocate_Pipe

The CICS TG does not perform a Deallocate_Pipe after it performs the initial ECI request from any given thread. This means that a pipe stays allocated for the lifetime of the thread and is not deallocated until one of the following occurs:

- An error is received by the CICS TG worker thread on an Open_Pipe call
- The CICS TG worker thread terminates
- The CICS TG address space terminates
- The CICS connection is placed out of service or IRC is closed
- The CICS region terminates

CICS TG thread usage

The Gateway daemon, used to receive network requests from remote Java clients, is itself a sophisticated multi-threaded Java application. It can handle multiple requests simultaneously and has a set of properties configured in the `CTG.INI` configuration file. Two pools of threads can be configured: the *connection manager* threads and the *worker* threads. For each connected client, one connection manager thread is used in the Gateway daemon, and is held until the client issues a disconnect using the `JavaGateway.close()` method, or is otherwise disconnected. In order for an ECI call to be performed via an allocated connection manager thread, a thread must be allocated from the worker thread pool for the duration of the ECI request. This relationship is summarized in Figure 7-6.

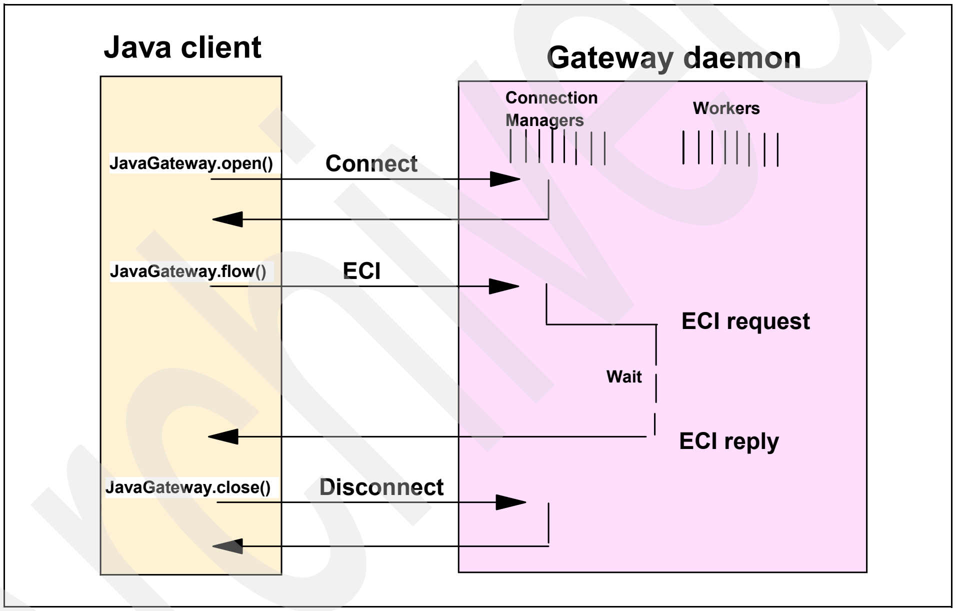


Figure 7-6 CICS TG threading model

Thus the connection manager threads limit the maximum number of connected Java clients, while the worker threads limit the number of concurrent ECI calls that can be issued by these attached clients. The initial and maximum numbers of these connection manager and worker threads are set in the `CTG.INI` file using the `maxconnect` and `maxworker` parameters. Requests from connection manager threads for worker threads can be queued internally, but will be timed out according to the `workertimeout` parameter.

EXCI pipe limitations

In CICS TS V1.3 and later, a single z/OS address space is limited (by the IRC code) to only being able to allocate up to 100 EXCI pipes in total, to all attached CICS regions. In contrast, maximum pipe usage in a CICS region is limited only by the number of sessions defined in the CICS MRO SESSIONS definition, which can be up to 999.

The following rules apply to the use of the EXCI by the CICS TG V5.0.0, and apply equally to worker threads within the Gateway daemon or to threads within WebSphere Application Server using the CICS TG local protocol.

- Each thread that services an ECI request will allocate an EXCI pipe. This pipe remains allocated until either the thread is terminated, or the CICS MRO connection is closed.
- Such pipes will be re-used by subsequent requests to the same worker thread, but pipes are not shared across different worker threads.
- If a worker thread connects to multiple CICS regions, then multiple EXCI pipes will be permanently allocated from the specific thread to these CICS regions.

If the CICS TG tries to allocate more pipes than there are available CICS sessions defined, the EXCI Open_Pipe call will fail with a RETRYABLE response, and a reason code 202. The ECI application will receive a return code -16 (ECI_ERR_RESOURCE_SHORTAGE) and the EXCI User replaceable module (DFHXCURM) will be invoked. In this instance you should consider configuring the RECEIVECOUNT parameter in the CICS SESSIONS definition to be at least 100.

If the number of EXCI pipes in use exceeds 100, the 101st EXCI Allocate_Pipe call will fail with a SYSTEM_ERROR response, and a reason code 608. The ECI application will receive a return code -9 (ECI_ERR_SYSTEM_ERROR). Since this means a limitation in the sending address space has been reached, you should consider reducing the number of threads (that is, `maxworker` in the Gateway daemon) to be less than or equal to 100/(number of attached CICS regions).

Restriction: If your Gateway daemon connects directly to multiple CICS regions, you will need to consider reducing the `maxworker` parameter to 100/(number of attached CICS regions). This is because in these circumstances a worker thread may allocate an EXCI pipe to multiple CICS regions.

To achieve workload distribution across multiple CICS AORs, it is therefore recommended that you implement a limited number of CICS listening regions (that connect to the CICS TG) and that these listening region workload balance and manage affinities to multiple AORs.

If the number of worker threads in your Gateway daemon becomes a limiting factor to increased throughput, we recommend you workload balance across cloned Gateway daemons, all listening on the same port using MVS TCP/IP port sharing. Each cloned Gateway daemon will be able to utilize up to 100 EXCI pipes. For further details on cloning Gateway daemons, refer to "Running multiple Gateway daemons" on page 168.

For further information on workload balancing using the CICS TG, refer to the redbook *Workload Management for Web Access to CICS*, SG24-6118.

7.3 Testing the configuration

To test our configuration we used the CICS TG sample Java applications, EciB1 and EciI1 from z/OS UNIX System Services and EciI1 from a Windows 2000 workstation. (The fourth character in EciI1 is an *I* as in *Intermediate*.) Both applications flow an ECI request to a connected CICS region through a CICS TG for z/OS (Figure 7-7). The CICS TG gateway and port are specified as input parameters.

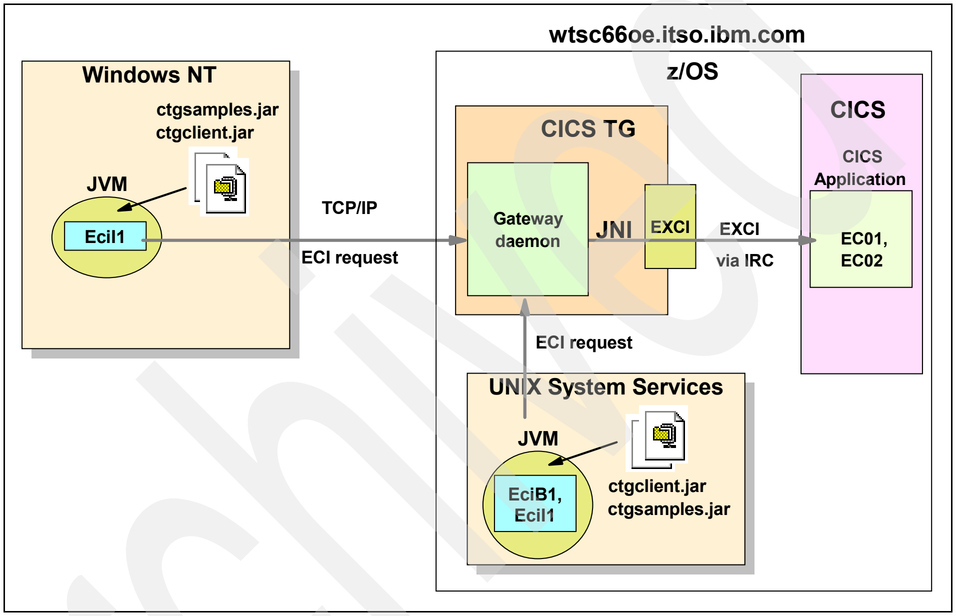


Figure 7-7 CICS TG for z/OS, software configuration

The EciB1 application tests basic communications capability to the CICS region. (The "B" stands for "basic".) The output of this application is simply the date and time as formatted by the CICS region.

The EciI1 application makes ECI calls in Extend_Mode. This causes the application to be transactional, allowing two-phase commit.

When called, the EciI1 application prompts the user for the target CICS region. Once that is input, the application starts a transaction, asking if they want to run again. When the user decides not to run the transaction, the application asks if the transaction should be committed or rolled back, performs the desired action, and ends.

We verified that our configuration was correctly set up as follows:

1. We started the CICS TG on z/OS as a started task from SDSF using the command **/S SCSCTG5**. Having started the job, we noticed this caused several temporary tasks to be invoked. These tasks were called SCSCTG5*n,* where *n* was a number from 1 to 9. However, once the CICS TG had started the original job SCSCTG5 was left running but paged out, and a child process SCSCTG55 was left running and paged in. SCSCTG55 is the UNIX Systems Services JVM process that is executing the Gateway daemon. This is shown in Example 7-9.

   To stop the CICS TG, we simply use the MVS **cancel** command against the original started task: **/C SCSCTG5**. Eventually, the second task will end as well, though you can cancel it too.

Example 7-9 CICS TG started tasks in SDSF

```
SDSF DA SC66  SC66     PAG    0 SIO   322 CPU   7/  6   DATA SET DISPLAYED
COMMAND INPUT ===>                                            SCROLL ===> CSR
NP   JOBNAME  StepName ProcStep JobID    Owner    C Pos DP Real Paging    SIO
     SCSCTG55 *OMVSEX           STC11097 CTGUSER    IN  F7  25T   0.00   0.42
     SCSCTG5  SCSCTG5  *OMVSEX  STC12191 CTGUSER    LO  FF  283   0.00   0.00
```

2. Once the CICS TG had started we checked the log file /ctg/scsctg5/logs/ctg.stdout for the message CCL6524I, indicating the TCP protocol handler had successfully started (Example 7-10).

Example 7-10 Successful start of CICS TG as seen in ctg.stdout

```
----------------------------------------------------------------
Set up environment variables for Java
----------------------------------------------------------------
PATH reset to /usr/lpp/java/IBM/J1.3/bin:/bin:.
JAVA_HOME reset to /usr/lpp/java/IBM/J1.3
CTG6111I File 'ctgenvvar' found. Using the configuration in script 'ctgenvvar'
to start up the application.
06/24/02 : 17:51:48:792 : CICS Transaction Gateway, Version 5.0.0, 5724-D12.
Build Level c000-20020621.
06/24/02 : 17:51:48:799 : (C) Copyright IBM Corporation 1999, 2002. All rights
06/24/02 : 17:51:48:946 : CCL8400I: Using ini file /ctg/scsctg5/CTG.INI.
06/24/02 : 17:51:48:973 : CCL6577I: Java version is 1.3.1.
06/24/02 : 17:51:49:012 : CCL6502I: Initial ConnectionManagers = 10, Maximum
ConnectionManagers = 90,
06/24/02 : 17:51:49:030 : CCL6502I: Initial Workers = 10, Maximum Workers = 90,
tcp: Port = 2006
06/24/02 : 17:51:49:038 : CCL6574I: Connection logging has been disabled.
06/24/02 : 17:51:50:431 : CCL6505I: Successfully created the initial
ConnectionManager and Worker threads.
06/24/02 : 17:51:53:828 : CCL6524I: Successfully started handler for the tcp:
protocol.
```

Tip: In V5 of the CICS TG, the standard logging messages no longer get sent to STDERR. Instead they are written to STDOUT and only trace and error messages are written to STDERR. In addition, connection logging messages are disabled by default, but can be enabled using the new parameter `connectionlogging` in the CTG.INI configuration file.

3. We checked the state of the TCP/IP sockets on our UNIX System Services TCP/IP stack using the **onetstat** command from OMVS. This showed us the output in Example 7-11 and indicates the CICS TG was indeed listening on port 2006 for TCP requests.

Example 7-11 OMVS onetstat

```
$ ÝSC66¨ /u/cicsrs2: onetstat
MVS TCP/IP onetstat CS V1R2      TCPIP Name: TCPIPOE         23:02:35
User Id  Conn     Local Socket           Foreign Socket        State
-------  ----     ------------           --------------        -----
SCSCTG55 0001F5C3 0.0.0.0..2006          0.0.0.0..0            Listen
```

4. We checked basic IP connectivity from our workstation to z/OS using the **ping** command from a Windows 2000 command prompt (Example 7-12).

Example 7-12 Ping to z/OS verifying hosts or DNS setup

```
C:\>ping wtsc66oe.itso.ibm.com

Pinging wtsc66oe.itso.ibm.com [9.12.6.29] with 32 bytes of data:

Reply from 9.12.6.29: bytes=32 time=120ms TTL=52
Reply from 9.12.6.29: bytes=32 time=110ms TTL=52

Ping statistics for 9.12.6.29:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 110ms, Maximum =  120ms, Average =  112ms
```

5. Next we performed our basic test under OMVS. First we used ISHELL to create /u/cicsrs2/EciB1Test. This will be an executable script file with the contents shown in Example 7-13.

Example 7-13 OMVS EciB1 test script file

```
export CLASSPATH=/usr/lpp/ctg500/ctg/classes/ctgsamples.jar
export CLASSPATH=$CLASSPATH:/usr/lpp/ctg500/ctg/classes/ctgclient.jar
echo $CLASSPATH
java com.ibm.ctg.samples.eci.EciB1 tcp://wtsc66oe.itso.ibm.com 2006
```

Once the file was created, we went into OMVS and marked EciB1Test as executable with the following command: **chmod +x EciB1Test**. You can verify this worked by doing a **ls -l** and verifying the last character in the first column is "x".

– You can see the first parameter after the name of the Java class is the location of the Gateway daemon (tcp://wtsc66oe.itso.ibm.com). Since the Gateway daemon was running on the same host as our Java client we could also have used the loopback IP address of 127.0.0.1 or "loopback" if it is defined in the TCP/IP configuration.

– The final parameter on the EciB1 line is the TCP/IP port used by the Gateway daemon. This defaults to 2006, but we added it anyway.

With the script marked as executable, we ran it by entering `EciB1Test`. The results of our tests are shown in Example 7-14.

Example 7-14 OMVS EciB1 test output

```
$ [SC66] /u/cicsrs2: EciB1Test
Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway Url] [Gateway Port Number]
Gateway tcp://wtsc66oe.itso.ibm.com port:2006

CICS Servers Defined:
        1. SCSCPJA1 -
        2. SCSCPJA2 -

Choose Server to connect to, or q to quit:
1

Program EC01 returned with data:-
       Hex: 32342f30362f30322031353a30363a32300
        ASCII text: 24/06/02 15:06:20
```

6. Once we knew the basic CICS TG functions were working, we tested the transaction EXCI functions using the supplied EciI1 sample class. We used ISHELL to create /u/cicsrs2/EciI1Test. This will be an executable script file with the contents shown in Example 7-15.

Example 7-15 OMVS test script file

```
export CLASSPATH=/usr/lpp/ctg500/ctg/classes/ctgsamples.jar
export CLASSPATH=$CLASSPATH:/usr/lpp/ctg500/ctg/classes/ctgclient.jar
echo $CLASSPATH
java com.ibm.ctg.samples.eci.EciI1 tcp://wtsc66oe.itso.ibm.com 2006
```

Once the file was created, we went into OMVS and marked EciI1Test as executable with the following command: **chmod +x EciI1Test**. You can verify this worked by using **ls -l** and verifying the last character in the first column is “x”.

Tip: Note to use the EciI1 sample you must make sure the Gateway daemon has access to the classes in ctgsamples.jar. To do this you will need to update the CTG_CLASSPATH environment variable in your ctgenvvar file. We added the following to the end of the current CTG_CLASSPATH definition

```
${CTG_CLASSES}/ctgsamples.jar
```

Without this class you will receive the error:

```
java.io.IOException: CCL6668E: Initial handshake flow failed.
[java.io.IOException: CCL6670E: Exception occurred in the Gateway.
[java.lang.ClassNotFoundException:com.ibm.ctg.samples.security.ServerComp
ression]]
```

With the script marked as executable, we ran it by entering `EciI1Test`. The results of our tests are shown in Example 7-16.

Example 7-16 OMVS EciI1 test output

```
CICSRS2 @ SC66:/u/cicsrs2>EciI1Test
Usage: java com.ibm.ctg.samples.eci.EciI1 ÝGatewayUrl¨ ÝPort¨
Gateway: tcp://wtsc66oe.itso.ibm.com:2006/

WTSC66OE.ITSO.IBM.COMGateway opened...

Number of systems = 2

-------------------------------------------------
Systems:
1: SCSCPJA1 -
2: SCSCPJA2 -
-------------------------------------------------

Select a server by number or 'Q' quit
1

Chosen server = SCSCPJA1
Waiting for a reply to a request to flow request
*
Reply received
No errors reported

-------------------------------------------------
ASCII:
        1st run OF EC02
```

```
Hex:
2020202020202020317374207275 6e204f46204543303220202020202020202020202020202020200
0000000000000000000000000000000000000000
-------------------------------------------------
Do you wish to run the program again in this Logical Unit of Work? - (Y/N)
N

C - Commit or R - rollback?
C
Committed logical unit of work on SCSCPJA1
Gateway closed
```

7. When we knew that basic and transactional EXCI was functional on z/OS, we tested the transactional functions in Windows 2000. We went into Windows and created a CMD file that contained the statements shown in Example 7-17. We set up a CMD file to make it easy to alter our classpath during testing.

   You will note the parameters passed to EciI1 includes the desired Gateway URL. In this case it is the hostname of our z/OS host. If you do not have DNS enabled, you could as easily include the TCP/IP address of the z/OS system or set up the name in your hosts file.

   The final parameter passed to EciI1 is the TCP/IP port that the Gateway daemon on z/OS is listening on. This defaults to 2006, but we specify it here for clarity.

Example 7-17 Windows test EciI1Test.cmd file

```
set CLASSPATH=.;C:\Program Files\IBM\IBM CICS Transaction Gateway\samples\java
set CLASSPATH=%CLASSPATH%;C:\Program Files\IBM\IBM CICS Transaction
Gateway\classes\ctgclient.jar
java com.ibm.ctg.samples.eci.EciI1 tcp://wtsc66oe.itso.ibm.com 2006
```

Example 7-18 shows the successful output from our Windows test.

Example 7-18 Windows EciI1 test

```
C:\EciI1Test.CMD
--------------------------------------------------
CICS Transaction Gateway Intermediate ECI Sample
--------------------------------------------------
Usage: java com.ibm.ctg.samples.eci.EciI1 [GatewayUrl] [Port]
Gateway: tcp://wtsc66oe.itso.ibm.com:2006/

WTSC66OE.ITSO.IBM.COMGateway opened...
Number of systems = 2

-------------------------------------------------
```

```
Systems:
1: SCSCPJA1 -
2: SCSCPJA2 -
-------------------------------------------------

Select a server by number or 'Q' quit
1

Chosen server = SCSCPJA1
Waiting for a reply to a request to flow request
*
Reply received
No errors reported
-------------------------------------------------
ASCII:
        1st run OF EC02
Hex:
20202020202020202031737420727 56e204f46204543303220202020202020202020202020202020200
0000000000000000000000000000000000000000
-------------------------------------------------
Do you wish to run the program again in this Logical Unit of Work? - (Y/N)
N

C - Commit or R - rollback?
C
Committed logical unit of work on SCSCPJA1
Gateway closed
```

7.4 Problem determination

In this section, we document information we learned while configuring this scenario and information on problem determination and tracing.

7.4.1 Tips and utilities

In this section, you will find useful commands and utilities for debugging problems with your configuration. You will also find some additional information about topics discussed in this chapter.

ECI return codes

The return codes for ECI calls can be found in the file header file cics_eci.h. This is not supplied on z/OS but can be found in the directory \ctg\include if using the

CICS TG on Windows, UNIX, or OS/2. A summary of some common errors we encountered is given below.

ECI_ERR_NO_CICS -3

We found this error can occur because of one of the following reasons:

- CICS is not started.
- Inter-region communication (IRC) has not been started.
- The NETNAME in your EXCI connection (if using a specific connection) does not match the value of your DFHJVPIPE variable referred to on the STDENV DD card in the CICS TG startup JCL.

ECI_ERR_SYSTEM_ERROR -9

This error generally implies that there is a problem with the EXCI. We found this can be caused because the CICS TG user ID can not log on to DFHIRP. This is controlled by the DFHAPPL.<*DFHJVPIPE*> and DFHAPPL.<*APPLID*> profiles in the RACF FACILITY class. For further details refer to Chapter 6, “CICS TG security scenarios” on page 99.

An ECI_ERR_SYSTEM_ERROR is also generated if you try to run a transactional EXCI application (such as EciI1) and do not have a valid CTG_RRMNAME specified.

An ECI_ERR_SYSTEM_ERROR can also be generated by security attach errors for the mirror transaction. For further details refer to page 129 in Chapter 6.

An ECI_ERR_SYSTEM_ERROR can also be generated if the number of EXCI pipes in use exceeds 100. For further details, refer to 7.2.3, “EXCI pipe usage” on page 154.

ECI_ERR_RESOURCE_SHORTAGE -16

This error can occur if the CICS TG tries to allocate more pipes than there are receive sessions defined in the CICS SESSIONS definition. In this circumstance, you should consider configuring the RECEIVECOUNT parameter in the CICS SESSIONS definition to be at least 100.

ECI_ERR_SECURITY_ERROR -27

This error can be caused by several different situations, including the following:

1. Surrogate security checking has failed. Check the MVS system console for RACF errors. For details on configuring surrogate security with the CICS TG for z/OS, refer to “Surrogate user security” on page 101.

2. The CICS TG failed to authenticate the user ID and password specified in the ECI call. If you do not wish to authenticate this user ID and password within the CICS TG, ensure the variable AUTH_USERID_PASSWORD in ctgstart is commented out (preceded by a # character).

STEPLIB — abend S806

We found that if the STEPLIB was set incorrectly in the shell of the user ID that ran the Gateway daemon, then the CICS TG would abend S806 because it was unable to find the module DFHXCPRX from the STEPLIB library CICSTS22.CICS.SDFHEXCI. This was indicated by the message:

```
CSV003I REQUESTED MODULE DFHXCPRX NOT FOUND
```

To circumvent this problem, you should ensure the STEPLIB is set correctly in the ctgenvvar file, using a statement such as:

```
EXCI_LOADLIB="CICSTS22.CICS.SDFHEXCI"
```

You should also check that the .profile of the user that runs the CICS TG does not set the STEPLIB to point to a different library that may contain conflicting modules.

z/OS TCP/IP commands

We found the following TCP/IP commands very useful when analyzing network problems with our CICS TG for z/OS:

- HOMETEST

  Issue this as a TSO command. If there is a valid TCP/IP address/host name, it will tell you what it is, along with other configuration information.

- NETSTAT

  Issue this command to get information about the TCP/IP sockets that are in use (TSO and other platforms). The UNIX System Services version of this command can be invoked using the **onetstat** command.

- NSLOOKUP <IP address>

  Issue this as a TSO command. A valid TCP/IP address will tell you the host name and vice versa.

- PING

  Use this to determine if an address is active and to resolve host names to IP addresses (see Example 7-19).

Example 7-19 Pinging the z/OS host

```
tso ping wtsc66oe.itso.ibm.com
EZA0458I Ping CS V1R2: Pinging host WTSC66OE.ITSO.IBM.COM  (9.12.6.29).
```

```
EZA0463I PING: Ping #1 response took 0.254 seconds. Successes so far 1.
```

Java version

To display the level of Java Development Kit (JDK) installed on your system, use the **`java -fullversion`** command as follows:

```
>java -fullversion
>java full version "J2RE 1.3.1 IBM OS/390 Persistent Reusable VM build
hm131s-20011206"
```

To find out what JVM is installed on your system, issue the following command to search for the location of the Java executable:

```
>type java
>java is /usr/lpp/java/IBM/J1.3/bin/java
```

If this reports that `java is not found`, then you will need to add the directory for the Java libraries to your PATH environment variable. This is best added in the USS /etc/profile so that all users can use Java.

USS SYS1.PARMLIB parameters

The BPXPRMxx parameters are used when z/OS UNIX System Services is initialized. The parameters that are currently in effect can be displayed by the command shown in Example 7-20.

Example 7-20 SYS1.PARMLIB(BPXPARxx)

```
D OMVS,OPTIONS
BPXO043I 19.34.33 DISPLAY OMVS 240
OMVS     000F ACTIVE          OMVS=(2C)
CURRENT UNIX CONFIGURATION SETTINGS:
MAXPROCSYS      =       4096    MAXPROCUSER     =      32767
MAXFILEPROC     =      65535    MAXFILESIZE     = NOLIMIT
MAXCPUTIME      = 2147483647    MAXUIDS         =        200
MAXPTYS         =        800
MAXMMAPAREA     =       4096    MAXASSIZE       = 2147483647
MAXTHREADS      =     100000    MAXTHREADTASKS  =      32767
MAXCORESIZE     =    4194304    MAXSHAREPAGES   =     331072
IPCMSGQBYTES    =     262144    IPCMSGQMNUM     =      10000
IPCMSGNIDS      =      20000    IPCSEMNIDS      =      20000
IPCSEMNOPS      =      32767    IPCSEMNSEMS     =      32767
IPCSHMMPAGES    =     524287    IPCSHMNIDS      =      20000
IPCSHMNSEGS     =       1000    IPCSHMSPAGES    =    2621440
SUPERUSER       = BPXROOT       FORKCOPY        = COW
STEPLIBLIST     = /usr/lpp/IMiner/steplib
USERIDALIASTABLE= /etc/ualiastable
PRIORITYPG VALUES: NONE
```

```
PRIORITYGOAL VALUES: NONE
MAXQUEUEDSIGS   =     100000    SHRLIBRGNSIZE    =   67108864
SHRLIBMAXPAGES  =    3145728    VERSION          = Z02RD1
SYSCALL COUNTS  = NO            TTYGROUP         = TTY
SYSPLEX         = YES           BRLM SERVER      = SC49
LIMMSG          = SYSTEM        AUTOCVT          = OFF
RESOLVER PROC   = DEFAULT
```

To make these parameter changes permanent, you must edit the BPXPRMxx member of your SYS1.PARMLIB library. The changes will take effect after the next IPL. You can also make dynamic changes by using the `SETOMVS` command. For example:

```
SETOMVS MAXPROCUSER=5000
```

In addition, BPXPRMxx also contains the mount commands for your HFS structure. To display which HFS directories are mounted (available), you can use the MVS `D OMVS, F` command as shown in Example 7-21.

Example 7-21 Partial listing of filesets mounted to UNIX System Services

```
D OMVS,F
BPXO045I 16.49.04 DISPLAY OMVS 526
OMVS     000F ACTIVE          OMVS=(2C)
TYPENAME   DEVICE ----------STATUS----------- MODE
NFS         23178 UNOWNED                     READ
  NAME=CDROM
  PATH=/SC42/cdrom
  MOUNT PARM=erprisc2:/cdrom,XLAT(Y),VERS(2)
  OWNER=         AUTOMOVE=N CLIENT=Y
HFS         25058 ACTIVE                      RDWR
  NAME=CTGUSER.SCSCTG5.HFS
  PATH=/ctg/scsctg5
  OWNER=SC66     AUTOMOVE=Y CLIENT=N
HFS         25058 ACTIVE                      RDWR
  NAME=CTGUSER.CTG500.HFS
  PATH=/usr/lpp/ctg500
  OWNER=SC66     AUTOMOVE=Y CLIENT=N
```

For more information on these settings, refer to *UNIX System Services Planning*, SC28-1890.

Running multiple Gateway daemons

Once you have CICS TG running, you may find it necessary to start a second Gateway daemon. There can be many reasons for this:

- You need test and production CICS TG regions

- You are upgrading to a new version
- You need more than 100 simultaneous pipes to CICS regions
- You need to workload manage incoming EXCI requests

The following areas need to be considered when cloning a CICS TG started task:

- Set RACF permissions on the CICS TG started task
- Clone started task JCL
- Create a new logging directory in UNIX System Services
- Create a new CONNECTION definition in CICS
- Create new STDENV member
  - Change the default CICS TG and TCP/IP ports
  - Change pipe name, if necessary
  - Change the location of the CTG.INI if necessary
  - Change the CICS TG starting location if necessary
  - Change the RRMNAME if necessary

We detail each step in this process. However, we assume you have previously performed these tasks to set up the gateway. We do not go into as much detail as we have in previous sections.

1. RACF permissions on the CICS TG started task

   If you have set up RACF definitions based on your CICS TG started task name, you will need to issue these commands again with your new CICS TG started task name. This could include creating a new user default user ID for your CICS TG started task.

   If you allow MRO link access on the CICS TG started task user ID, you will have to update this as well.

2. Clone started task JCL in your proclib

   This involves creating a new started task and pointing it to a new STDENV file. You will also want to point your STDOUT and STDERR to different locations.

3. Create a new logging directory in UNIX System Services

   The default is to send logs and traces to the same directory as the CICS TG executables. We found that turning traces on could consume a great deal of space.

   In our CICS TG JCL, we point our traces to a different directory, which is more of a good administration practice than a requirement. We mounted a different HFS data set as this directory, so our logging directory could fill up without causing problems for the CICS TG.

4. New connection on CICS

While it is possible to share a pipe, it’s best to set up a new connection within CICS. We found it helpful in problem diagnosis to set up a different pipe as well as a different terminal prefix. Any transactions coming in via the new pipe are assigned the new terminal ID.

5. Create a new STDENV member

   a. Change the default CICS TG and TCP/IP ports

      Depending on your implementation, you might want to change the default ports. If you do this, be sure to also change the applications connecting to the new CICS TG.

      If you are cloning your CICS TG to support workload balancing, you may want to keep the same ports and use the SHAREPORT parameter of TCP/IP.

   b. Change pipe name, if necessary

      Changing the pipe name when cloning a CICS TG may be a good idea, because it can help with diagnostics. Doing this will also require a new CICS CONNECTION definition and changes to the RACF profiles that control MRO link security.

   c. Change the location of the CTG.INI if necessary

      We used the CICSCLI variable to set the location of our CTG.INI to our own /ctg/scsctg4 HFS directory. If you run multiple Gateway daemons, we suggest you set up a new CTG.INI file for each Gateway, unless you wish to clone address spaces when using TCP/IP port sharing.

   d. Change the CICS TG starting location if necessary

      If cloning the same release and maintenance level, you do not have to update the stating location of the CICS TG in the CICS TG JCL. However, if you are testing new maintenance or a different release, you will have to change the starting locations using the environment variable CTGSTART_HOME. For details on how we set our environment variables, refer to 7.2.1, “Defining CICS TG environmental variables” on page 148.

      You may make references to the starting location in your ctgenvvar and STDENV DD members, so you should check these as well.

   e. Change the CTG_RRMNAME if necessary

      If using RRS, you will need to set up a unique CTG_RRMNAME for each CICS TG address space.

7.4.2 Tracing

To demonstrate the types of traces available, we show you how to diagnose an error using EXCI, CICS TG and JNI traces. We created an error by specifying an incorrect CTG_RRMNAME (CCL.IBM.CTG5.XX) as the CICS TG environment variable. This caused an ECI_ERR_SYSTEM_ERROR (-9) return code on the ECI call.

Tip: The error in our CTG_RRMNAME variable was that it did not end in UA. For further details on using transactional EXCI and defining the CTG_RRMNAME, refer to 7.2.1, “Defining CICS TG environmental variables” on page 148.

EXCI trace

The first trace we illustrate is an EXCI trace of the Gateway daemon started task on z/OS. This trace is more complicated to set up than the other traces, but has the advantage that the trace table is stored in the Gateway region address space, and so can be analyzed after the error (providing the trace is still in the buffer).

The process for getting an EXCI trace is outlined in the manual *z/OS Gateway Administration*, SC34-5935, and is repeated here for clarity.

1. Use the following command to display all OMVS tasks:

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

2. Find the Gateway address space, and note the address space ID (ASID).

Tip: By default the Gateway started task uses two address spaces, one for the UNIX System Services shell running the ctgstart script and one for the address space running the JVM for the Gateway daemon. In our configuration, the ctgstart address space was called SCSCTG5 and the JVM address space was called SCSCTG55. We dumped the JVM address space SCSCTG55, because this is the address space that performs EXCI calls.

3. Enter the DUMP command with a suitable comment. For example:

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

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

   ```
   /R 495,ASID=401,END
   ```

   where *495* is the message number for the reply, and *401* is the ASID.

5. The system will dump as shown in Example 7-22. Note the dump name created. In our example, the dump name was DUMP.D0529.H17.SC66.#MASTER#.S00072.

Example 7-22 Dumping the Gateway address space in SDSF

```
R 495,ASID=401,END
IEE600I REPLY TO 495 IS;ASID=401,END
```

```
DUMPID=022 REQUESTED BY JOB (*MASTER*)
DUMP TITLE=JGATE DUMP RRS
$HASP100 BPXAS    ON STCINRDR
$HASP373 BPXAS    STARTED
IEF403I BPXAS - STARTED - TIME=13.26.12 - ASID=008A - SC61
IEF196I IGD101I SMS ALLOCATED TO DDNAME (SYS00076)
IEF196I         DSN (DUMP.D0529.H17.SC66.#MASTER#.S00072        )
IEF196I         STORCLAS (SCDUMP) MGMTCLAS (STANDARD) DATACLAS (
IEF196I )
IEF196I         VOL SER NOS= DUMPS2
```

6. Go into IPCS and select option 0. Enter the dump name as shown in Figure 7-8 on page 172.

```
----------------------- IPCS Default Values ---------------- LOCAL updated
Command ===>

  You may change any of the defaults listed below.  The defaults shown before
  any changes are LOCAL.  Change scope to GLOBAL to display global defaults.

  Scope   ==> LOCAL   (LOCAL, GLOBAL, or BOTH)

  If you change the Source default, IPCS will display the current default
  Address Space for the new source and will ignore any data entered in
  the Address Space field.

  Source  ==> DSNAME('DUMP.D0529.H17.SC66.#MASTER#.S00072')
  Address Space   ==>
  Message Routing ==> NOPRINT TERMINAL
  Message Control ==> CONFIRM VERIFY FLAG(WARNING)
  Display Content ==> MACHINE REMARK REQUEST STORAGE SYMBOL

Press ENTER to update defaults.
```

Figure 7-8 Defining the dump to IPCS

7. Format the dump using the CICS TS V2.2 trace formatter by going to option 6 within IPCS and entering `VERBX DFHPD620 'TR=1'` (the last parameter requests an abbreviated trace). Pressing Enter on this screen formats the dump. IPCS asks whether you want to use summary dump data. Reply **Y** as shown in Figure 7-9.

```
IKJ56650I TIME-01:29:19 PM. SERVICE-2194025 SESSION-01:28:36 MAY 29,2002
BLS18122I Init in progress for DSNAME('DUMP.D0529.H17.SC66.#MASTER#.S00072')
BLS18124I TITLE=JGATE DUMP RRS
BLS18223I Dump written by z/OS 01.02.00 SVC dump - level same as IPCS level
BLS18222I z/Architecture mode system
BLS18160D May summary dump data be used by dump access?
Enter Y to use, N to bypass.
Y
BLS18123I 43,332 blocks, 180,261,120 bytes, in
DSNAME('DUMP.D0529.H17.SC66.#MASTER#.S00072')
IKJ56650I TIME-01:29:44 PM. CPU-00:00:05 SERVICE-2393143 SESSION-01:29:00 MAY
Dump of z/OS 01.02.00 - level same as IPCS level
***
```

Figure 7-9 Formatting the Gateway dump in IPCS

8. On the resulting IPCS formatted dump output, you now need to find the EXCI trace table. To do this, enter the command `FIND 'TRACE TABLE'`. Example 7-23 show the output from our job. Note the `ECI_PARAMETERS_OUTPUT` line. The return code of -9 equates to an ECI_ERR_SYSTEM_ERROR, which is a generic code for many types of EXCI failures.

Example 7-23 IPCS dump trace table

```
INTERNAL TRACE TABLE

XCI   EXCI  EX 1001 XCPRH EXIT  INITIALIZE_USER       OK,OK
XCI   EXCI  EX 8000 JVDLL ENTRY ECI_PARAMETERS_PASSED Worker-0,1
XCI   EXCI  EX 8003 JVDLL DATA  CONVERTED_PARAMETER   Worker-0,jstrServer,SCSCPJA1
XCI   EXCI  EX 8003 JVDLL DATA  CONVERTED_PARAMETER   Worker-0,jstrProgram,EC02
XCI   EXCI  EX 8004 JVDLL DATA  INBOUND_COMMAREA      Worker-0,80,11517C78
XCI   EXCI  EX 8030 ????? ????? **_POINT_ID_NOT_RECOGNISED_**
XCI   EXCI  EX 8007 JVDLL EXIT  ECI_PARAMETERS_OUTPUT Worker-0,1,-9
XCI   EXCI  EX 03ED ????? ????? **_POINT_ID_NOT_RECOGNIZED_**
XCI   EXCI  EX 8000 JVDLL ENTRY ECI_PARAMETERS_PASSED Worker-0,1
```

However, at this stage the EXCI trace doesn't give us enough information to solve the error, so we continue on to the Gateway daemon trace.

Tip: You can also use the MVS generalize trace facility (GTF) to analyze both CICS and EXCI traces. However, this tracing is better suited to in-depth error analysis, since it requires the GTF proc to be started and initialized in order to collect the data. It does, however, provide the very useful feature of being able to interleave both EXCI and CICS trace entries. For further details on how we used GTF tracing, refer to "GTF tracing" on page 179.

Gateway daemon trace

The Gateway daemon trace shows calls that the CICS TG sends and receives. There are two ways of starting the Gateway daemon trace, dynamic and static. Dynamic tracing requires the new TCPAdmin protocol handler be set up in the CTG.INI file. The static trace depends on parameters specified in the CICS TG startup JCL and requires the CICS TG address space to be recycled.

Dynamic trace (gateway)

Here we assume the TCPAdmin protocol handler has been activated in the CTG.INI file. This allows an administrator to activate and de-activate trace on a running Gateway using the supplied ctgadmin.jar file. To activate the TCPAdmin protocol handler, we added the following lines to our CTG.INI file:

```
protocol@admin.handler=com.ibm.ctg.server.RestrictedTCPHandler
     protocol@admin.parameters=port=2810;\
                               sotimeout=1000;\
                               connecttimeout=2000;\
                               idletimeout=600000;\
                               pingfrequency=60000;\
                               maxconn=5;
```

1. From within TSO OMVS, we changed our directory to /usr/lpp/ctg500/ctg/classes/. We entered the command to query the current state of the trace:

   ```
   java -jar ctgadmin.jar -ctg=tcp:\\wtsc66oe.itso.ibm.com:2810 -a=qtrace
   ```

   Example 7-24 shows the result of this command. The tlevel in the Gateway trace settings is 0, indicating tracing is off.

Example 7-24 CICS TG for z/OS TCPAdmin query command

```
CTGCtrl - CTG Control Program, version 5.0.0
(C) Copyright IBM Corporation 2002. All rights reserved
Gateway trace settings:
  tlevel=0
  truncationsize=80
  dumpoffset=0
  tfile=/ctg/scsctg5/logs/ctg.trace
  tfilesize=0

JNI trace settings:
  tlevel=0
  tfile=/ctg/scsctg5/logs/jni.trace
The command completed successfully
```

2. To turn trace on, we enter:

   ```
   java -jar ctgadmin.jar -ctg=tcp:\\wtsc66oe.itso.ibm.com:2810
   -a=setgwtrace -tlevel=4
   ```

We see the output:

```
CTGCtrl - CTG Control Program, version 5.0.0
(C) Copyright IBM Corporation 2002. All rights reserved
The command completed successfully
```

3. To verify our command worked, we enter the query command again:

```
java -jar ctgadmin.jar -ctg=tcp:\\wtsc66oe.itso.ibm.com:2810 -a=qtrace
```

Example 7-25 CICS TG for z/OS TCPAdmin query command

```
CTGCtrl - CTG Control Program, version 5.0.0
(C) Copyright IBM Corporation 2002. All rights reserved
Gateway trace settings:
  tlevel=4
  truncationsize=80
  dumpoffset=0
  tfile=/ctg/scsctg5/logs/ctg.trace
  tfilesize=0

JNI trace settings:
  tlevel=0
  tfile=/ctg/scsctg5/logs/jni.trace
The command completed successfully
```

Static trace (gateway)

1. On your CICS TG JCL you first need to modify the start procedure to specify the use of tracing and a trace file. In our example, the CICS TG runs as a started task. The command is similar but is specified as a JCL PARM as shown below:

```
//GATEV500 EXEC PGM=BPXBATCH,REGION=0M,
//  PARM='SH /usr/lpp/ctg500/ctg/bin/ctgstart -noinput -x
//             -tfile=/ctg/scsctg5/logs/ctg.trace'
```

2. After the CICS TG task is stopped and started, the sample is attempted again and the resulting trace output file is viewed. In Example 7-26 we see the CICS TG makes a call to the JNI. The `Before JNI` line shows the specifics of the call, while the `After JNI` line shows the return code (-9) resulting from the JNI call.

   The last line shows that the CICS TG takes the response from the JNI, and formats the ECI_ERR_SYSTEM_ERROR message to send to the CICS TG.

Example 7-26 CICS TG trace table

```
13:40:50:880 : ConnectionManager-0: S-C: CCL6602I: GatewayRequest type = ECI, flow version =
4194304, flow type = 1, Gateway return code = 0, length of data following the header = 41.
13:40:50:950 : ConnectionManager-0: S-C: CCL6513I: About to receive remainder of a request.
Ýcom.ibm.ctg.server.ServerECIRequest¨
```

```
13:40:51:040 : ConnectionManager-0: S-C: CCL6727I: ECIRequest: Commarea_Length = 80.
13:40:51:063 : ConnectionManager-0: S-C: CCL6720I: ECIRequest: Call_Type = ECI_ASYNC, Server =
SCSCPJA1, Program = EC02, Transid = , Extend_Mode = ECI_EXTENDED, Luw_Token = 0,
Message_Qualifier = 0, Callbackable = true.
13:40:51:069 : ConnectionManager-0: S-C: CCL6727I: ECIRequest: Commarea_Length = 80.
13:40:51:151 : ConnectionManager-0: S-C: CCL6512I: Waiting to receive a request.
ÝConnectionManager-0¨
13:40:51:166 : Worker-0: S-C: CCL6514I: Accepting work request. ÝWorker-0¨
ÝConnectionManager-0¨
13:40:51:389 : Worker-0: .TCPHandler:CCL6603I: # Dump: 32/32 bytes : Offset = 0 Reply flow
13:40:51:410 : Worker-0: S-C: CCL6523I: About to execute work request. ÝWorker-0¨
13:40:51:428 : Worker-0: S-C: CCL6695I: Before JNI CcicsECI(1) : Server=SCSCPJA1, Program=EC02,
Commarea.Length=80, ExtendMode=1, LUW=0.
13:40:51:514 : Worker-0: S-C: CCL6693I: After JNI CcicsECI(1) : Rc=-9, LUW/TermIndex=0
13:40:51:581 : Worker-0: S-C: CCL6721I: ECIRequest: Call_Type = ECI_ASYNC, Cics_Rc =
ECI_ERR_SYSTEM_ERROR, Abend_Code = , Luw_Token = 0, Message_Qualifier = 0.
```

This trace tells us the CICS TG is working normally but we need to look further into the JNI. The next step is to the trace the CICS TG JNI module (libCTGJNI) to determine what is causing the bad return code from the JNI call.

JNI trace

The third type of trace is the CICS TG Java Native Interface (JNI) trace. Like the CICS TG trace, there are two ways of starting the JNI trace, dynamic and static.

Dynamic JNI tracing is new in CICS TG for z/OS V5.0 and requires the TCPAdmin be set up in the CTG.INI. We demonstrate dynamic tracing, but used static tracing for our problem diagnosis.

The static trace depends on parameters specified in the CICS TG startup JCL and requires the CICS TG address space to be recycled.

New in CICS TG for z/OS V5.0 is the default logging of EXCI return codes. These are written to unique files in the directory $HOME/ibm/ctgjnilog.* and are very useful for quick problem determination.

Dynamic trace (JNI)

Here we assume the TCPAdmin client has been specified in the CTG.INI file.

1. From within TSO OMVS, we changed our directory to /usr/lpp/ctg500/ctg/classes/. We entered the command to query the current state of the trace:

   ```
   java -jar ctgadmin.jar -ctg=tcp:\\wtsc66oe.itso.ibm.com:2810 -a=qtrace
   ```

   Example 7-27 shows the result of this command. The tlevel in the JNI trace settings is 0, indicating tracing is off.

Example 7-27 CICS TG for z/OS TCPAdmin query command

```
CTGCtrl - CTG Control Program, version 5.0.0
(C) Copyright IBM Corporation 2002. All rights reserved
Gateway trace settings:
  tlevel=4
  truncationsize=80
  dumpoffset=0
  tfile=/ctg/scsctg5/logs/ctg.trace
  tfilesize=0

JNI trace settings:
  tlevel=0
  tfile=/ctg/scsctg5/logs/jni.trace
The command completed successfully
```

2. To turn trace on, we entered:

```
java -jar ctgadmin.jar -ctg=tcp:\\wtsc66oe.itso.ibm.com:2810
-a=setjnitrace -tlevel=1
```

We saw the output:

```
CTGCtrl - CTG Control Program, version 5.0.0
(C) Copyright IBM Corporation 2002. All rights reserved
The command completed successfully
```

3. To verify our command worked, we entered the query command again:

```
java -jar ctgadmin.jar -ctg=tcp:\\wtsc66oe.itso.ibm.com:2810 -a=qtrace
```

Example 7-28 CICS TG for z/OS TCPAdmin query command

```
CTGCtrl - CTG Control Program, version 5.0.0
(C) Copyright IBM Corporation 2002. All rights reserved
Gateway trace settings:
  tlevel=4
  truncationsize=80
  dumpoffset=0
  tfile=/ctg/scsctg5/logs/ctg.trace
  tfilesize=0

JNI trace settings:
  tlevel=1
  tfile=/ctg/scsctg5/logs/jni.trace
The command completed successfully
```

Static trace (JNI)

This is activated using the CTG_JNI_TRACE environment variable, which can be set in your ctgstart script, in the ctgenvvar file, or in the STDENV DD in the CICS

TG started task. We chose the STDENV method and added the following parameter to the PDS member referenced on our STDENV DD statement:

```
CTG_JNI_TRACE=/ctg/scsctg5/logs/jni.trace
```

This caused the JNI trace to be written out to the HFS file specified. The JNI trace includes information about the ECI requests and the EXCI flows the CICS TG creates. Example 7-29 shows the output of the JNI trace.

Example 7-29 CICS TG trace table

```
CICS Transaction Gateway JNI Trace file for z/OS Version 5.0 Service Level 00 , Build Level
c000-20020621
04:30:50.745 CM-0        ¨ : CCL6806I: CcicsInit: Register with RRS. Return code = 768.
04:30:53.172 Worker-0    ¨ : CCL6800I: CcicsECI: ECI parameters on entry.  Call_Type = 1,
Extend_Mode = 1, Luw_Token = 0, Commarea_Length = 80, Cics_Rc = 0, Flags = 0.
04:30:53.190 Worker-0    ¨ : CCL6801I: CcicsECI: Performed parameter conversion.  parameter
name = jstrServer, parameter value = SCSCPJA1.
04:30:53.206 Worker-0    ¨ : CCL6801I: CcicsECI: Performed parameter conversion.  parameter
name = jstrProgram, parameter value = EC02    .
04:30:53.219 Worker-0    ¨ : CCL6802I: CcicsECI: Input commarea information.  commarea length =
80, commarea address = 11517c78.
04:30:53.234 Worker-0    ¨ : CCL6818E: CcicsECI: Begin Context failed. RRM Return code = 1878.
04:30:53.248 Worker-0    ¨ : CCL6805I: CcicsECI: ECI parameters on exit.  Call_Type = 1,
Extend_Mode = 1, Luw_Token = 0, Commarea_Length = 80, Cics_Rc = -9, AV = 0.
```

We begin reading our trace from the bottom up. The last line shows the output we pass back to the CICS TG. The next line up shows we received a return code 1878 from RRS.

In looking at the *MVS Programming: Resource Recovery,* GC28-1739, we see that error codes must be converted from decimal to hexadecimal. Decimal 1878 is hex 756, which translates to CRG_AUTH_FAILURE. The text of a 756 error tells us the program encountered an error registering with the resource manager. At this point, we should start to suspect RRS and thus the setting of the CICS TG CTG_RRMNAME variable.

A further confirmation of this lies in the second line from the top. This shows the CICS TG trying to register its RRMNAME with RRS. It receives a 768 return code, which is a hex 300. The *MVS Programming: Resource Recovery,* GC28-1739 shows the 300 error to be a CRG_RM_NAME_INV and the help text of the message details that the resource manager name specified in the call is incorrect.

Knowing this information, we changed the CTG_RRMNAME to a valid name (CCL.CTG.IBM.UA), restarted the CICS TG, and the example started working.

Tip: In CICS TG for z/OS V5.0 the EXCI JNI logs can also now be quickly and easily used to identify the majority of EXCI problems. The logs are written to unique files in the directory $HOME/ibm/ctgjnilog.

EXCI subreasons

Finally, we created an error to illustrate the use of EXCI subreasons. These subreasons are helpful in that they provide another facet of problem diagnosis.

This time we forced an error by closing IRC on the CICS region. The EXCI and Gateway traces simply provide a generic EXCI 203 (ECI_ERR_NO_CICS) error. The 203 error also shows up on the JNI trace, so we skip the EXCI and Gateway traces.

We ran our test application and Example 7-30 shows the significant line from the JNI trace.

Example 7-30 JNI trace table, EXCI subreasons

```
01:05:11.721 Worker-0    ¨ : CCL6822E: CcicsECI: EXCI function error.  Function
Call = 3, Response = 8, Reason = 203, Subreason field-1 = 92,
subreason field-2 = 0, Cics_Rc = -3
```

We see the 203 error that we would have seen on the EXCI trace table and the Gateway trace. In addition, we see a Subreason field-1 of 92.

To find out the meaning of a subreason 92, we need to look in CICSTS22.CICS.SDFHMAC(DFHIRSDS). After converting decimal 92 to a hex 5C, we see this equates to “SYSTEM NOT LOGGED ON”.

This is more specific than the general NO_CICS error and led us to check to see the state of IRC on our target CICS region. Of course we knew this, since we deliberately closed it in the first place.

GTF tracing

Both the EXCI and CICS itself support trace entries to be written to GTF. In the following example, we give a brief explanation of how we debugged a simple problem with the CICS TG for z/OS using GTF tracing. In this error scenario, the call using EciB1 failed with ECI_ERR_TRANSACTION_ABEND (-7) and AEI0, due to the disabling of the invoked program, EC01 within CICS.

Note: The principal advantage of GTF tracing is that the EXCI and CICS trace entries can be combined in the same output, which can help considerably in problem determination. However, the Gateway daemon and JNI trace cannot be written to GTF and must be analyzed separately.

- To start with, you must enable both EXCI and CICS trace entries to be written to GTF. The DFHXCOPT table is used to control EXCI tracing and to activate an EXCI trace. You need to specify TRACE=2, and GTF=ON in the DFHXCOPT table, as shown in Example 7-31.

Example 7-31 EXCI options table, DFHXCOPT

```
DFHXCO TYPE=CSECT,
               TIMEOUT=0,              No timeout
               TRACE=2,                Trace entries
               TRACESZE=1024,          Trace table
               DURETRY=30,             Retry SDUMPS for 30 seconds
               TRAP=OFF,               DFHXCTRA - OFF
               GTF=ON,                 GTF - ON
               MSGCASE=MIXED,          Mixed case messages
               CICSSVC=0,              EXCI will obtain CICS SVC number
               CONFDATA=SHOW,          Show user commarea data in trace
               SURROGCHK=YES           Perform surrogate-user check @P1C
         END   DFHXCOPT
```

- GTF tracing from CICS can be activated using the CETR transaction as shown in Figure 7-10, by setting the GTF Trace Status to STARTED. We also reduced the CICS trace entries to IS=1-2 and PC=1 to capture only MRO and program control trace entries. This can be achieved using PF4 on the CETR screen.

```
CETR              CICS Trace Control Facility              PJA1 SCSCPJA1

Type in your choices.

Item                              Choice      Possible choices

Internal Trace Status      ===>  STARTED     STArted, STOpped
Internal Trace Table Size  ===>  64      K   16K - 1048576K

Auxiliary Trace Status     ===>  STOPPED     STArted, STOpped, Paused
Auxiliary Trace Dataset    ===>  A           A, B
Auxiliary Switch Status    ===>  NEXT        NO, NExt, All

GTF Trace Status           ===>  STARTED     STArted, STOpped

Master System Trace Flag   ===>  ON          ON, OFf
Master User Trace Flag     ===>  ON          ON, OFf

When finished, press ENTER.



PF1=Help      3=Quit      4=Components      5=Ter/Trn     9=Error List
```

Figure 7-10 CICS CETR transaction

- To activate GTF, you will need to start your GTF started task. The JCL for our started task is shown in Example 7-32.

Example 7-32 GTF proc

```
//GTFNEW  PROC MEMBER=GTFPARM
//IEFPROC EXEC PGM=AHLGTF,PARM='MODE=EXT,DEBUG=NO,TIME=YES',
//  TIME=1440,REGION=2880K
//IEFRDER DD   DSNAME=SYS1.TRACE, UNIT=SYSDA,SPACE=(CYL,100),
//             DISP=SHR
//SYSLIB  DD   DSNAME=SYS1.PARMLIB(&MEMBER),DISP=SHR
```

- The SYS1.PARMLIB(GTF) member referenced in Example 7-32 on page 181 defines the following options for GTF:

```
TRACE=USRP
USR=F6C
END
```

- Once GTF has initialized, you will need to reply U to the initialization message as shown in Example 7-33.

Example 7-33 Reply to GTF commands

```
17.40.18 STC08965 *002 AHL125A  RESPECIFY TRACE OPTIONS OR REPLY U
17.40.30 STC08965  R 002,U
17.40.30 STC08965  U
17.40.30 STC08965  AHL906I THE OUTPUT BLOCK SIZE OF     4096 WILL BE USED FOR
OUTPUT  111
  111                DATA SETS:
  111                 WAKELIN.TRACE
17.40.30 STC08965  AHL080I GTF STORAGE USED FOR GTF DATA:  112
  112               GTFBLOCK STORAGE     40K BYTES (BLOK=      40K)
  112                PRIVATE STORAGE    1024K BYTES
  112               SADMP HISTORY       40K BYTES (SADMP=     40K)
  112               SDUMP HISTORY       40K BYTES (SDUMP=     40K)
  112               ABEND DUMP DATA      0K BYTES (ABDUMP=     0K)
17.40.30 STC08965  AHL031I GTF INITIALIZATION COMPLETE
17.40.57 STC08965  AHL006I GTF ACKNOWLEDGES STOP COMMAND
```

- GTF tracing is now active and trace entries from the CICS TG and the CICS region will be captured by GTF.
- Once the trace is complete you should terminate the CICS TG started task using the command `/P GTF`.
- To format GTF trace entries, you will need to enter IPCS from SDSF and then from option 6 enter the command:

  `GTF DSN('SYS1.TRACE') USR`

  Where `SYS1.TRACE` is your GTF trace data set allocated in your GTF proc (see Example 7-32 on page 181).

  This will then produce the formatted trace output for the EXCI and CICS trace entries. An abbreviated output from the formatted trace is shown in Example 7-34, showing the ECI return code -7 caused by the our PGMIDERR condition.

Example 7-34 Formatted GTF EXCI and CICS trace entries

```
EX 8003 JVDLL  DATA  - CONVERTED_PARAMETER JAVA_TID(Worker-0) NAME(jstrServer) VALUE(SCSCPJA1)
....
EX 8003 JVDLL  DATA  - CONVERTED_PARAMETER JAVA_TID(Worker-0) NAME(jstrProgram) VALUE(EC01    )
....
EX 1000 XCPRH  ENTRY - OPEN_PIPE TOKEN(1B503130) TO CICS(SCSCPJA1) BY USER(SCSCTG5 )
....
EX 2001 XCPRH EVENT IRP_CONNECT TO CICS(SCSCPJA1) BY USER(SCSCTG5 )
....
EX 1001 XCPRH  EXIT - OPEN_PIPE   RESPONSE(OK)  REASON(OK)
```

```
....
EX 1000 XCPRH  ENTRY - DPL TO PROGRAM(EC01   ) ON CICS(SCSCPJA1) USING PIPE(1B503130) BY
USER(SCSCTG5 )
....
EX 2004 XCPRH EVENT SWITCH_REQUEST SENT TO CICS(SCSCPJA1) BY USER(SCSCTG5 )
....
EX 2005 XCPRH DATA DATA SENT ON PIPE(1B503130) BY USER(SCSCTG5 )
....
AP DD18 CRNP  EVENT - DEQUEUE WORK ELEMENT TYPE (NEW CONNECT WITH INPUT RECEIVED) TIMESTAMP
(B7F888FFBCB65E06)  SCCB AT 7F58B SYSTEM SCSCTG5
....
EX 0802 XCPRH  *EXC* - PGMIDERR_DETECTED - PROGRAM(EC01   ) ON CICS(SCSCPJA1) BY USER(SCSCTG5 )
....
EX 8011 JVDLL  *EXC* - DPL_REQUEST_ERROR ECI_RC(-7)
....
EX 1000 XCPRH  ENTRY - CLOSE_PIPE TOKEN(1B503130) TO CICS(SCSCPJA1) BY USER(SCSCTG5 )
....
EX 2002 XCPRH EVENT IRP_DISCONNECT FROM CICS(SCSCPJA1) BY USER(SCSCTG5 )
```

8

SSL connections to the Gateway daemon on z/OS

In this chapter, we show you how we implemented an encrypted Secure Sockets Layer (SSL) connection from a remote Java client application to our CICS Transaction Gateway (CICS TG) running on z/OS. We used both the System SSL protocol handler and the Java Secure Sockets Extension (JSSE) SSL protocol handler. We tested the configuration using the sample Java application EciB1 running as a stand-alone Java application on a Windows workstation. This is illustrated in Figure 8-1 on page 186.

For details on how to install the CICS TG on z/OS and configure the Gateway daemon, refer to Chapter 7, “TCP connections to the Gateway daemon on z/OS” on page 133.

For more details on the SSL protocol, refer to Chapter 3, “Security technologies” in *Securing Web Access to CICS*, SG24-5756.

8.1 Preparation

The following section details the software levels we used in our sample configuration and gives you a checklist of the definitions required to begin your configuration.

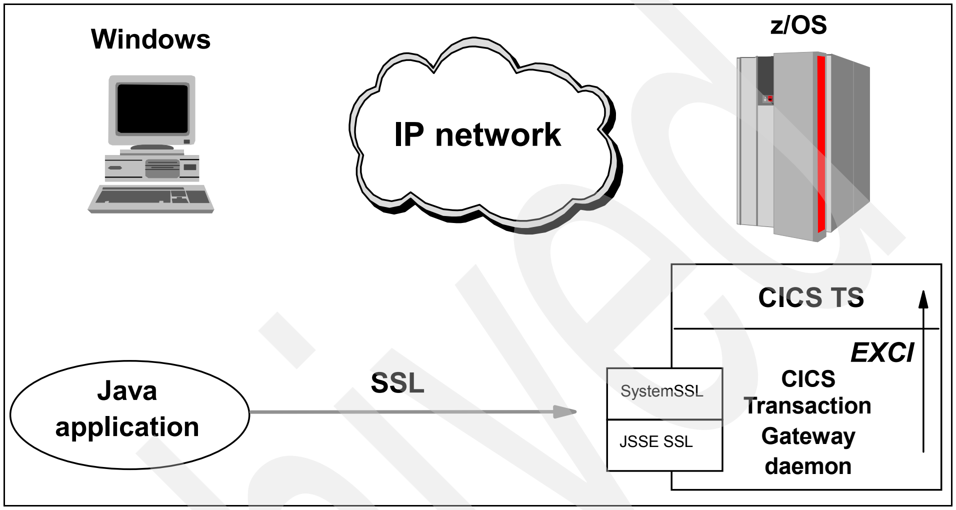


Figure 8-1 Software components: CICS TG for z/OS with SSL

In the following section, you will find details on how we:

- Configured Windows to use the JSSE libraries.
- Obtained an externally signed SSL server test certificate and deployed it for use with the CICS TG using the z/OS tool gskkyman.
- Configured a self-signed server certificate and deployed it for use with the CICS TG using the Java tool keytool.
- Created a JSSE client keystore.
- Configured the Java client on Windows to use the CICS TG Java classes.

8.1.1 Software checklist

We used the levels of software shown in Table 8-1.

Table 8-1 Software checklist: CICS TG for z/OS SSL

Client Workstation	z/OS
WIndows 2000 Service Pack 2	z/OS V2.1
CICS Transaction Gateway V5.00	CICS Transaction Gateway V5.00
IBM Java Development Kit V1.3.0	IBM Java Development Kit V1.3.1S
	CICS Transaction Server V2.2
JSSE Version 1.0.2 build 20020411.	JSSE included in JDK
JCE Version 1.2.1 build 020118.	

8.1.2 Definitions checklist

The definitions we used to configure this scenario are summarized in Table 8-2 and Table 8-3 on page 188. The definitions we used for creating SSL certificates are listed in Table 8-4 on page 188. Before you configure your components, we recommend that you document the same parameters for your configuration.

Table 8-2 Definitions checklist: CICS TG for z/OS SSL

CICS TG for z/OS	CICS Transaction Server	Example
hostname		wtsc66oe.itso.ibm.com
CICS TG network protocol		ssl
System SSL port		8052
JSSE SSL port		8062
pipe	NETNAME	SCSCTG5
	APPLID	SCSCPJA1
SSL key database and keystore password		default
SSL key label and keystore alias		ITSO

Table 8-3 Definitions checklist: CICS TG for z/OS SSL JDK configuration

Platform	JDK install directory
Windows	C:\Program Files\IBM\Java13
z/OS	/usr/lpp/java/IBM/J1.3

Table 8-4 Definitions checklist: Certificate field information

X.500 distinguished name field	gskkyman field	Example
cn	common name	wtsc66oe.itso.ibm.com
o	organization	IBM
ou	organization unit	ITSO
l	city/locality	San Jose
s	state/province	California
c	country name	US

8.2 Configuration

There are three versions of SSL that you can use with the z/OS CICS TG.

- System SSL
- SSLight
- JSSE

We will discuss these in the following sections.

SystemSSL

System SSL is specific to z/OS and uses the native SSL function provided by the z/OS Integrated Cryptographic Service Facility (ICSF). It offers the added advantage of being able to utilize the z/OS Cryptographic Coprocessor feature, which can substantially reduce the CPU cost of SSL handshakes and SSL data encryption. It is supported by its own key management tool, which is known as *gskkyman*, and can use both externally signed and self-signed certificates. Certificates created and managed by System SSL are held in a key database file (.kdb).

SSLight

The CICS TG supports the Java-based SSLight toolkit, which is provided with CICS TG on all platforms (including z/OS). It is supported by the CICS TG key management tool ctgikey, which uses the iKeyman tool; iKeyman uses Java class files (termed *keyrings*) to hold certificates.

JSSE

The CICS TG also supports the Java-based JSSE library that is provided with CICS TG on all platforms (including z/OS). It is supported by the JSSE key management tools keytool and iKeyman. JSSE uses files (termed *keystores*) to hold certificates. Keytool provides a command-line based interface for manipulating keystores, whereas iKeyman uses a GUI interface. Keytool only generates X.509 Version 1 certificates, whereas iKeyman generates X.509 Version 3 certificates, which are more flexible than older versions.

Although you have a choice of using System SSL, SSLight, or JSSE on z/OS, only JSSE or SSLight can be used by the CICS TG Java classes on the workstation (the Java client). System SSL is restricted for use by a CICS TG protocol handler only.

We chose to use JSSE to test with, since it supports a higher level of security than the SSLight toolkit. JSSE supports a length of 128 bits for the keys used to encrypt network data, whereas SSLight supports a maximum length of 56 bits. Because the strength of an encryption is related to the length of the keys used, 128-bit is significantly stronger than 56-bit encryption. An additional consideration is that support for SSLight might be removed in a future CICS TG release.

To make the Gateway daemon and CICS TG Java class libraries use JSSE, the appropriate libraries need to be present in the JDK, and a JSSE keystore is specified as the key file to use, rather than an SSLight keyring class.

In order to set up a secure SSL connection to our CICS TG on z/OS, we performed the following steps.

Configuring z/OS for JSSE

Because the necessary JSSE libraries were already present in the JDK, we did not have to modify our z/OS JDK.

Configuring Windows for JSSE

We had to install the JSSE library into the JDK on our Windows workstation. We also had to install the IBM Java Cryptographic Extension (JCE) provider into the JDK. We did this as follows:

1. We took the two JSSE zip files supplied with the CICS TG product (ibm-jsse.zip and ibm-jce.zip) and copied them into our JDK directory.

2. Using WinZip, we extracted the contents of these zip files into the JDK directory C:\Program Files\IBM\Java13 (Figure 8-2).

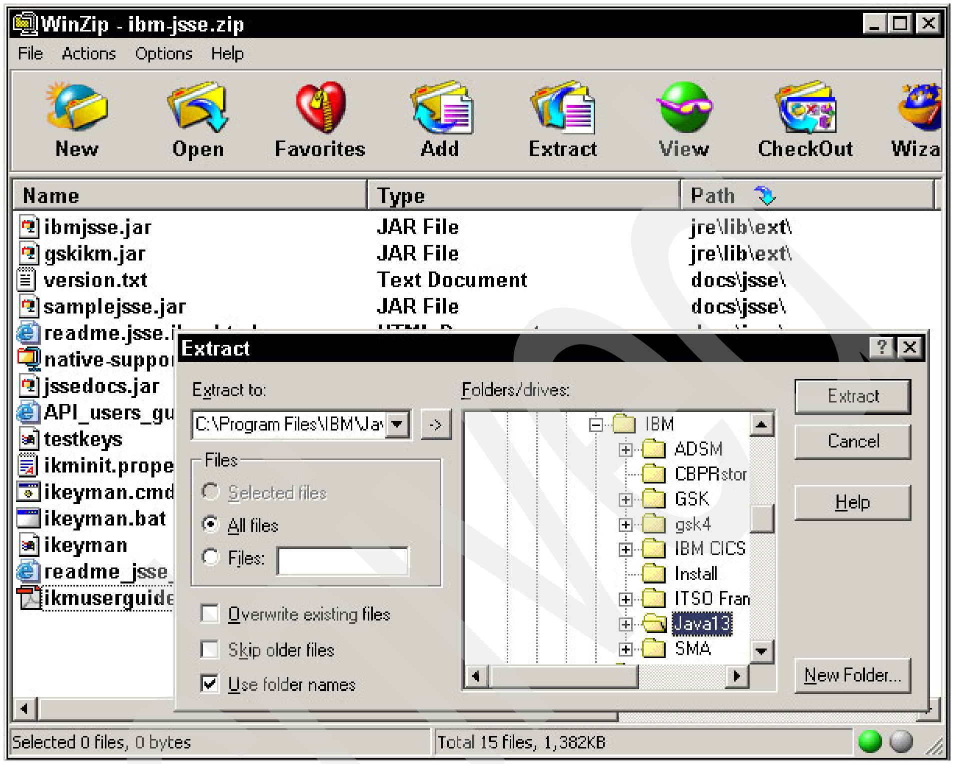


Figure 8-2 WinZip extracting the JSSE library into the JDK

3. We selected the **Yes to All** button when the Confirm File Overwrite window appeared.

4. We modified the JDK master security properties file java.security located in the directory C:\Program Files\IBM\Java13\jre\lib\security to add the IBM JCE provider to the JDK, as shown below. Note that we placed the IBM JCE provider after the Sun provider that is available with the JDK.

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.ibm.crypto.provider.IBMJCE
```

8.2.1 Configuring the server certificate

We configured a server key database for use with System SSL and a server keystore file for use with JSSE. System SSL used a test certificate from a certificate authority, whereas JSSE used a self-signed certificate.

System SSL

For System SSL it was necessary to create a server key database, request an externally signed test certificate, and receive this certificate into our key database.

Creating a key database on z/OS

We performed the following steps to create our key database on z/OS:

1. In a z/OS UNIX System Services shell we changed directory to the HFS directory /web/scsctg5 that we were using for our CICS TG for z/OS installation, and entered the **gskkyman** command to invoke the IBM Key Management Utility.
2. From this menu, we selected **Option 1** (**Create new key database**), as shown in Example 8-1.

Example 8-1 Creating a new key database

```
IBM Key Management Utility

Choose one of the following options to proceed.

   1  - Create new key database
   2  - Open key database
   3  - Change database password

   0  - Exit program

Enter your option number: 1
```

3. We continued by replying to the following prompts, as shown in Example 8-2.

Example 8-2 Entering details of the new key database

```
Enter key database name or press ENTER for "key.kdb": systemssl.kdb
Enter password for the key database.......>
Enter password again for verification.....>
Should the password expire? (1 = yes, 0 = no) [1]: 0

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

For more details on System SSL and the gskkyman utility, refer to *OS/390 V2R10.0 System SSL Programming Guide and Reference*, SC24-5877.

Requesting an externally signed test certificate

Once we had created our key database, we performed the following steps to request an externally signed test certificate.

1. In order to obtain an externally signed certificate from a certificate authority (CA), we first had to generate a certificate signing request (CSR). We chose **Option 3 (Create New Key Pair and Certificate Request)** from the Key database menu and entered the definitions in Table 8-4 on page 188, as shown in Example 8-3.

Example 8-3 Creating a new key pair and certificate request

```
Enter option number (or press ENTER to return to the parent menu): 3
Enter certificate request file name or press ENTER for "certreq.arm":
Enter a label for this key................> ITSO
Select desired key size from the following options (512):
    1:    512
    2:    1024
Enter the number corresponding to the key size you want: 2
Enter certificate subject name fields in the following.
    Common Name (required)................> wtsc66oe.itso.ibm.com
    Organization (required)...............> IBM
    Organization Unit (optional)..........> ITSO
    City/Locality (optional)..............> San Jose
    State/Province (optional).............> California
    Country Name (required 2 characters)..> US

Please wait while key pair is created...

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

> **Important:** When creating the certificate request in gskkyman, we found that we needed to specify all fields including the optional ones. We could not use an abbreviation for the state/province; otherwise VeriSign rejected our certificate signing request.

2. After the certificate had been generated, we checked the contents of the file (Example 8-4) using the following UNIX System Services command:

   `cat certreq.arm`

Example 8-4 Contents of the certificate request file

```
$ [SC66] /ctg/scsctg5: cat certreq.arm
-----BEGIN NEW CERTIFICATE REQUEST-----
```

```
MIIBLDCB1wIBADByMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTER
MA8GA1UEBxMIU2FuIEpvc2UxDDAKBgNVBAoTA0lCTTENMAsGA1UECxMESVRTTzEe
MBwGA1UEAxMVd3RzYzY2b2UuaXRzby5pYm0uY29tMFwwDQYJKoZIhvcNAQEBBQAD
SwAwSAJBAMmrTVGvmBpIH2WFB4sbgptMzydLSuMpRsFPyr9uYhmSr0VDi7Vpa4MU
UC3WjMvn/48KJt1sV4Kvl+ZUFbKMK5OCAwEAAaAAMA0GCSqGSIb3DQEBBAUAA0EA
JQXsNdj2o9EZpa5QhjsVZO2nr9eYbWzlLTN2Oj8rVpW4HLudmZxAMys8mnwPjz8g
BtV6UHUQZvh0rQSO4h1WnQ==
-----END NEW CERTIFICATE REQUEST-----
```


3. The next step was to use this file to request an externally signed server certificate from our chosen CA. We chose to use VeriSign's test facility, and applied online for our certificate at their Web site http://www.verisign.com. Using this method it was merely necessary to fill in an online form, and paste in our CSR. The following is a summary of the process we had to go through:
   - The first window required us to fill in personal information, such as name, address, phone number, e-mail address, need for Web security, and your responsibility for Web security in your company.
   - An enrollment window was then displayed with recommendations about what to do before starting. Following this we were guided through a five-step process:
     i. Generate CSR (we had already done this step).
     ii. Submit CSR.

        This window asked us to cut and paste the CSR file we had created in the Base64 Encoded ASCII format. We found the most reliable way was to FTP our CSR request to Windows and use the Notepad Editor to cut and paste the CSR into the HTML form. Without this we found extra characters were copied in the operation, resulting in VeriSign reporting an `fffffffe` error.
     iii. Complete application data.

        This window displayed the information extracted from the CSR request we had entered, and asked for information about the person to whom the certificate was to be sent. We submitted our application and were informed that VeriSign was processing our request.
     iv. After a few hours we received an e-mail containing the Base64 Encoded test certificate, signed using the private key from the VeriSign Test CA root certificate and valid for 14 days. This also contained instructions about the next two steps: installing the test CA root and installing the test server ID. However, these steps were not necessary in our situation since we were using a Java client instead of a Web browser and the CICS Transaction Gateway instead of a Web server.

Note: We found that VeriSign would only issue one certificate for a server, so we were unable to obtain a test certificate to use with our SSL protocol handler.

Receiving our test certificate

Once we received our test certificate, we copied and pasted this into a file (verisigncert.arm) on our workstation using the Notepad Editor (being very careful to remove any trailing space characters), and then transferred this to the HFS directory /web/scsctg5 on z/OS using FTP in ASCII mode. We then received this into our key database on z/OS using Option 4 of the Key Database Menu - Receive a certificate issued for your request.

Example 8-5 Receiving the certificate issued for our request

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

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

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

JSSE

For JSSE it was necessary to create a server keystore and create a self-signed certificate. We used the keytool to complete these steps.

Creating a keystore on z/OS using keytool

We performed the following steps to create our keystore on z/OS using keytool:

1. In a z/OS UNIX System Services shell we changed the directory to the HFS directory /ctg/scsctg5 we were using for our z/OS CICS TG.
2. We used the **keytool** command in Example 8-6 to invoke the Key Management Tool to create a keystore with a self-signed certificate in the file jssesslss.jks.

Example 8-6 Using keytool to create a keystore with a self-signed certificate

```
keytool -genkey -alias ITSO -keysize 1024
-dname "cn=wtsc66oe.itso.ibm.com,o=IBM,ou=ITSO,l=San Jose,s=California,c=US"
-keystore jssesslss.jks -keypass default -storepass default -keyalg RSA
```

The options on the **keytool** command are as follows:

genkey Generates a key pair and wraps the public key into a self-signed certificate.

alias Stores the self-signed certificate and private key in a new keystore entry identified by `ITSO`.

dname Specifies the X.500 distinguished name to be associated with the alias. This is used as the issuer and subject fields of the self-signed certificate. The distinguished name consists of a number of fields separated by commas.

keystore The keystore location.

keypass The password used to protect the private key.

storepass The password used to protect the integrity of the keystore.

keyalg Specifies the algorithm to be used to generate the key pair.

The keytool we used generated a self-signed certificate and private key using a distinguished name that specified the same information as our System SSL certificate request. The values we used for each field are listed in Table 8-4 on page 188. We specified the RSA algorithm to generate our key pair. We specified the same password for both the keystore and the private key.

3. To see the self-signed certificate we had created, we used the **`keytool`** command in Example 8-7 to invoke the Key Management Tool to list the contents of the keystore. We used the `-v` parameter to show all details of the certificate in the keystore. Example 8-7 shows that the keystore has one entry and our certificate is owned by the same distinguished name as it is issued by.

Example 8-7 Using keytool to list the contents of a keystore

```
$[SC66] /ctg/scsctg5: keytool -list -v -keystore jssesslss.jks -storepass
default

Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry:

Alias name: itso
Creation date: Tue Jun 25 17:10:35 EDT 2002
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=wtsc66oe.itso.ibm.com, O=IBM, OU=ITSO, L=San Jose, ST=California,
C=US
Issuer: CN=wtsc66oe.itso.ibm.com, O=IBM, OU=ITSO, L=San Jose, ST=California,
C=US
Serial number: 3d18dc4a
```

```
Valid from: Tue Jun 25 17:10:34 EDT 2002 until: Mon Sep 23 17:10:34 EDT 2002
Certificate fingerprints:
         MD5:  6D:5B:C0:14:12:E5:E2:47:C1:8E:FE:B8:D5:9D:08:36
         SHA1: AA:8D:BF:D5:74:88:E8:72:D9:83:3D:12:E3:03:F3:2E:54:97:74:58

*******************************************
*******************************************
```

8.2.2 Configuring the client keyring

For the client side, we used the JSSE library to provide an SSL connection.

In order to test the System SSL protocol, we needed to create a JSSE keystore that contains the signer certificate for the VeriSign Test Certificate Authority who signed our server certificate, obtained in 8.2.1, “Configuring the server certificate” on page 191. This is because the client needs the public key of the signer, contained inside the signer certificate, to decrypt the server certificate presented to the client during the SSL handshake. If the server certificate is decrypted successfully, then the client can trust that the certificate is authentic and SSL communication can occur.

To test the JSSE SSL protocol, we needed to add our self-signed server certificate to this JSSE keystore. The client needs to have the signer certificate of our server to recognize the certificate the SSL protocol will present to it during the SSL handshake.

Client keystore for SystemSSL and JSSE

To create a keystore for use with SystemSSL and JSSE, we used the iKeyman tool, provided with the JSSE libraries, on Windows. We performed the following steps:

1. From a Windows command prompt we changed into the CICS TG `bin` directory with the command:

   `cd C:\Program Files\IBM\IBM CICS Transaction Gateway\bin`

2. From a Windows command prompt, we started the iKeyman tool using the command:

   `java com.ibm.ikeyman.Ikeyman`

   The initial window is shown in Figure 8-3 on page 197.

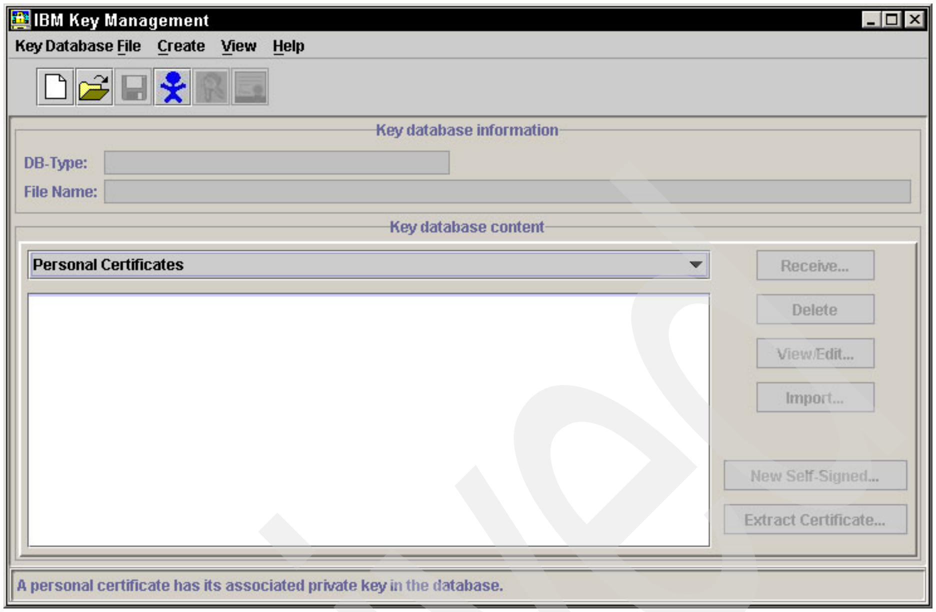


Figure 8-3 iKeyman initial window

3. We created a new keystore using **Key Database File -> New**. This caused the New window to appear (Figure 8-4). We made sure the Key database type was JKS and entered `jsseclientss.jks` in the File Name field. The Location field was already set to the CICS TG bin directory.

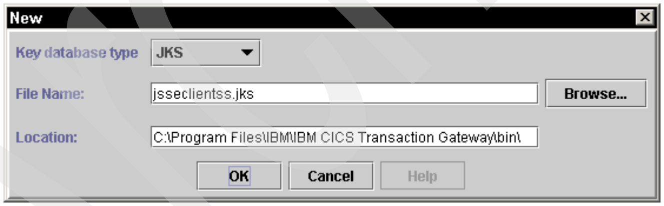


Figure 8-4 iKeyman new keystore window

4. We clicked **OK** and were presented with the password prompt window. We entered the password `default`.

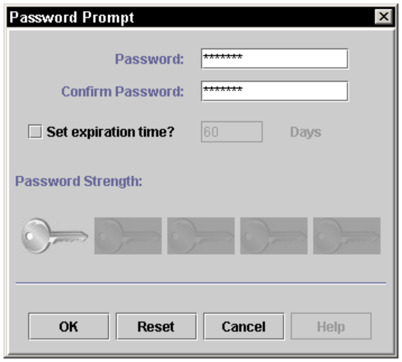


Figure 8-5 iKeyman password prompt window

5. We clicked **OK** and were presented with the Signer Certificates window as shown in Figure 8-6.

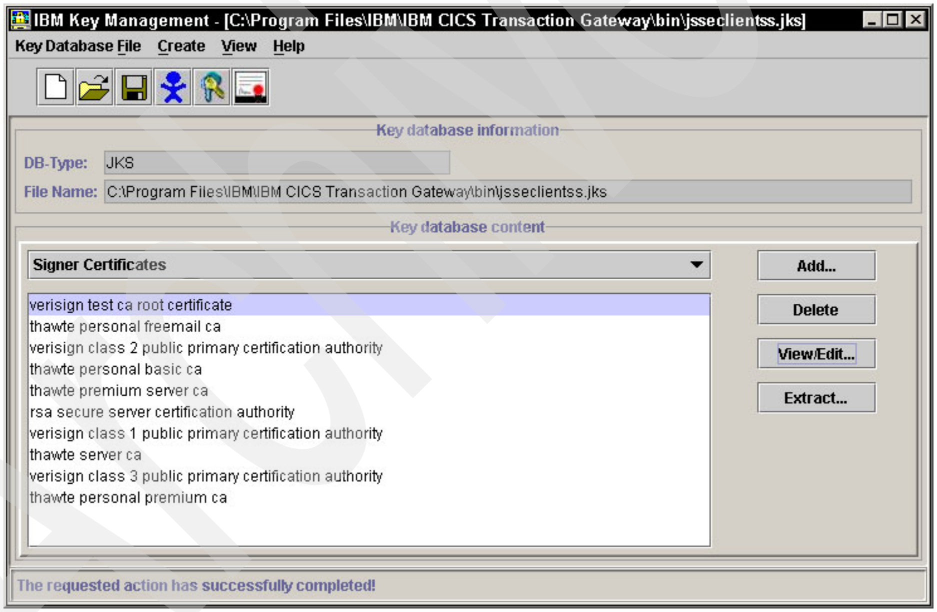


Figure 8-6 iKeyman signer certificates window

You can see that the VeriSign Test CA root certificate is supplied by default in a new keystore as a signer certificate. Therefore, we did not need to import this into the keystore.

Tip: We found that iKeyman does not remember the directory the file window was in the last time it is used, but always opens the directory from where iKeyman was launched. Therefore we changed to the CICS TG bin directory before launching iKeyman so that the file window would open there.

To add the signer certificate of our self-signed server certificate to the client keystore, we performed the following steps:

1. From a z/OS UNIX System Services shell, we changed to the /ctg/scsctg5 directory where our server keystore was.
2. We used the **keytool** command in Example 8-8 to invoke the Key Management Tool to export the self-signed certificate from the server keystore in the file jssesslss.jks into a file called server.der.

Example 8-8 Exporting the self-signed certificate

```
$ [SC66] /ctg/scsctg5: keytool -export -alias ITSO -file server.der
-keystore jssesslss.jks -storepass default
Certificate stored in file <server.der>
```

3. We transferred the server.der file from z/OS to the C:\Program Files\IBM\IBM CICS Transaction Gateway\bin directory on our Windows workstation using FTP. We used binary mode for the transfer.
4. From the Signer Certificates window in iKeyman we clicked the **Add** button. This caused the Add CA's Certificate from a File window to appear (Figure 8-7). We changed the Data type to `Binary DER data`. We changed the Certificate file name field to `server.der`.

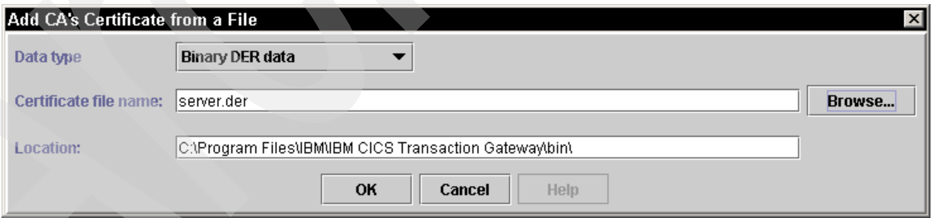


Figure 8-7 iKeyman add CA certificate window

5. We clicked the **OK** button. This displayed the Enter a Label window (Figure 8-8 on page 200). We entered `ITSO ca root certificate` into the window and clicked the **OK** button.

Figure 8-8 iKeyman enter certificate label window

6. The certificate was imported successfully and we were able to view it by selecting it in the Signer Certificates window and clicking the **View/Edit** button (Figure 8-9).

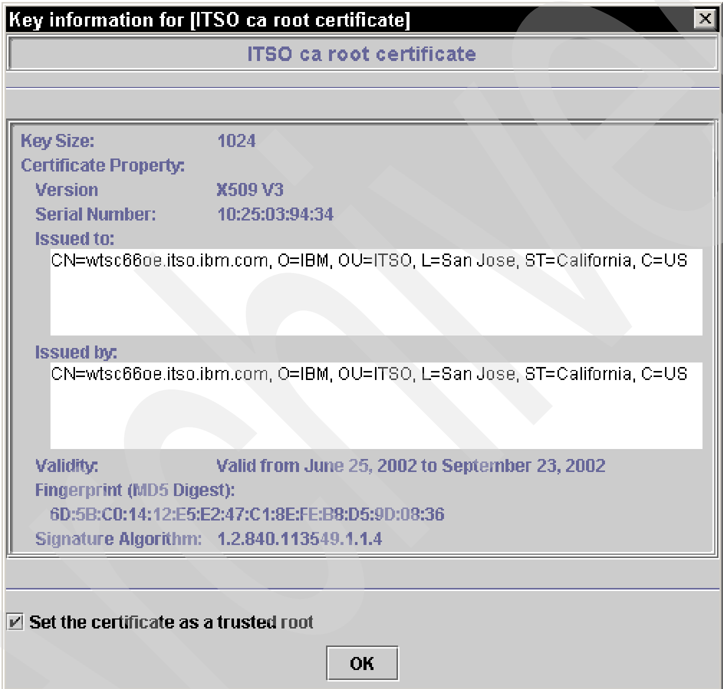


Figure 8-9 iKeyman viewing the self-signed certificate

As can be seen, the certificate is issued to the same distinguished name as it is issued by. It has also been set as a trusted root in the keystore.

> **Tip:** We found it difficult to tell the difference between the CICS TG version of iKeyman and the JSSE version. The only differences we could spot were the blue icon on the toolbar of the JSSE iKeyman, which allowed a new provider to be added, and the JSSE version did not have a Recreate Request button on the Personal Certificates window.

8.2.3 Configuring the CICS TG for SSL

We had to configure our CICS TG to enable System SSL and JSSE SSL.

System SSL configuration

We performed the following steps to configure our CICS TG to be able to use our SystemSSL server certificate:

1. When using System SSL, binaries and data sets for the product code must be marked as program controlled, as must any key database file (.kdb). We performed this using the following **extattr** commands:

   ```
   extattr +p /usr/lpp/gskssl/lib/*
   extattr +p /usr/lpp/gskssl/bin/*
   extattr +p /web/scsctg5/systemssl.kdb
   ```

   Before issuing **extattr** commands, we required Resource Access Control Facility (RACF) access to the BPX.FILEATTR.PROGCTL profile. This was obtained by using the following command:

   ```
   PERMIT BPX.FILEATTR.PROGCTL CLASS(FACILITY) ID(CICSRS3) ACCESS(READ)
   SETROPTS RACLIST(FACILITY) REFRESH
   ```

   We verified that the files were program controlled using the **ls -E** command from OMVS. The second set of attributes are the extended ones. The second column of these should contain the character “p”.

2. We modified our CICS TG configuration file (/ctg/scsctg5/CTG.INI) to activate the SystemSSL protocol handler to use our SystemSSL server certificate, as shown in Example 8-9.

Example 8-9 Enabling the SystemSSL protocol handler

```
protocol@systemssl_ssl.handler=com.ibm.ctg.server.GskSslHandler
protocol@systemssl_ssl.parameters=port=8052;sotimeout=1000;\
connecttimeout=2000;idletimeout=600000;pingfrequency=60000;\
keyring=/ctg/scsctg5/systemssl.kdb;keyringpw=default;clientauth=off;
```

You can see that the CICS TG is configured to listen on port 8052 for SSL requests, and has access to systemssl.kdb, where we stored our server certificate.

SSL client authentication

It is also possible to associate an SSL client certificate with a RACF user ID. This is useful if you wish to use SSL client authentication to identify your CICS TG user, as opposed to a RACF user ID and password. We did not do this in our test scenario, but to do this you need to run the TSO command `RACDCERT`. The certificate supplied from the CA is used to create a matching profile in the RACF CLASS(DIGTCERT). The syntax of `RACDCERT` is:

```
RACDCERT ADD('CICSRS3.CERT.ARM') TRUST ID(CICSRS3)
```

Where `CICSRS3.CERT.ARM` is a z/OS sequential file of Virtual Basic (VB) format that has been copied from the public key certificate (in our case the Hierarchical File System [HFS] file verisigncert.arm) using the OMVS copy command. When you issue the `RACDCERT` command, RACF creates a digital certificate profile in CL(DIGTCERT) with TRUST status that associates a certificate with a user ID. This profile can then be used to translate a certificate into a user ID. For more information, see *OS/390 SecureWay™ Security Server RACF Command Language Reference,* SC28-1919.

JSSE SSL configuration

We performed the following steps to configure our CICS TG to be able to use our JSSE SSL server certificate:

1. We modified the CICS TG configuration file (/ctg/scsctg5/CTG.INI) to activate the SSL protocol handler to use our JSSE server keystore.

Example 8-10 Enabling the SSL protocol handler

```
protocol@ssl.handler=com.ibm.ctg.server.SslHandler
protocol@ssl.parameters=connecttimeout=2000;idletimeout=60000;\
      keyring=/ctg/scsctg5/jssesslss.jks;keyringpw=default;\
      pingfrequency=60000;port=8062;solinger=0;sotimeout=1000;
```

You can see that the CICS TG is configured to listen on port 8062 for SSL requests, and has access to jssesslss.jks, where we stored our self-signed server certificate. Because we specified a keystore file in the `keyring` parameter, the CICS TG will use JSSE to provide SSL support.

8.3 Testing the configuration

To test our configuration, we used the CICS TG sample Java application EciB1. This is installed in the CICS TG samples directory, inside the java subdirectory. The sample's Java class is defined to be in the Java package `com.ibm.ctg.samples.eci`, so the sample source file is stored under the following directory structure:

```
com/ibm/ctg/samples/eci
```

The full path to the EciB1 sample on our Windows system is:

```
C:\Program Files\IBM\IBM CICS Transaction Gateway\samples\java\
  com\ibm\ctg\samples\eci\EciB1.java
```

The sample is also provided in a compiled form inside ctgsamples.jar.

We used EciB1 from our Windows 2000 workstation to call the CICS program EC01 via SystemSSL and JSSE. The EciB1 application flows an ECI request to a connected CICS region through a specified Gateway (Figure 8-10) and invokes the program EC01. The Gateway URL is specified as an input parameter. The CICS region is entered at the interactive prompt.

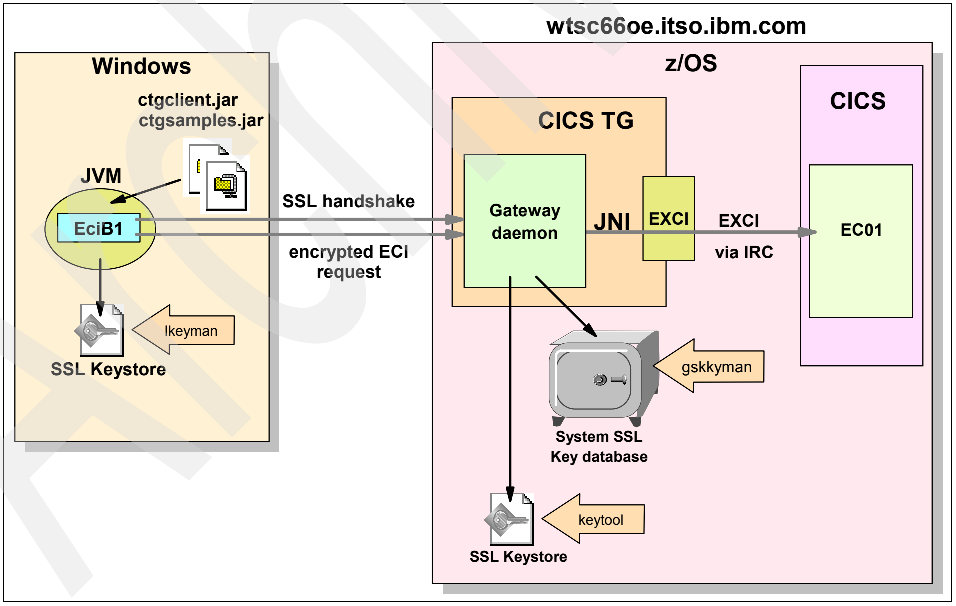


Figure 8-10 CICS TG for z/OS SSL software configuration for remote testing

After we installed and configured the software components as illustrated in Figure 8-10, we tested our configurations as follows:

1. We started the CICS TG on z/OS as a started task from SDSF using the command `/S SCSCTG5`. Once the CICS TG had started the original job, SCSCTG5 was left running but paged out, and a child process SCSCTG55 was left running and paged in. SCSCTG55 is the CICS TG daemon that is running in UNIX System Services. This is shown in Example 8-11.

Example 8-11 CICS TG started tasks in SDSF

```
SDSF DA SC66  SC66     PAG    0 SIO   322 CPU   7/  6   DATA SET DISPLAYED
COMMAND INPUT ===>                                            SCROLL ===> CSR
NP   JOBNAME  StepName ProcStep JobID    Owner    C Pos DP Real Paging    SIO
     SCSCTG55 *OMVSEX           STC11097 CTGUSER    IN  F7  25T   0.00   0.42
     SCSCTG5  SCSCTG5  *OMVSEX  STC12191 CTGUSER    LO  FF  283   0.00   0.00
```

2. Once the CICS TG had started, we checked the log file /ctg/scsctg5/logs/ctg.stdout for the message CCL6524I, indicating the System SSL and SSL protocol handlers had successfully started (see Example 8-12).

Example 8-12 Successful start of the CICS TG as seen in ctg.stdout

```
CTG6111I File 'ctgenvvar' found. Using the configuration in script 'ctgenvvar'
to start up the application.
06/26/02 : 12:44:23:257 : CICS Transaction Gateway, Version 5.0.0 Pre-release,
5724-D12. Build Level c000-20020621.
06/26/02 : 12:44:23:260 : (C) Copyright IBM Corporation 1999, 2002. All rights
reserved.
06/26/02 : 12:44:23:354 : CCL8400I: Using ini file /ctg/scsctg5/CTG.INI.
06/26/02 : 12:44:23:356 : CCL6577I: Java version is 1.3.1.
06/26/02 : 12:44:23:357 : CCL6502I: Initial ConnectionManagers = 10, Maximum
ConnectionManagers = 90,
06/26/02 : 12:44:23:359 : CCL6502I: Initial Workers = 10, Maximum Workers = 90,
tcp: Port = 2006
06/26/02 : 12:44:23:360 : CCL6574I: Connection logging has been disabled.
06/26/02 : 12:44:24:166 : CCL6505I: Successfully created the initial
ConnectionManager and Worker threads.
06/26/02 : 12:44:26:779 : CCL8402I: JSSE libraries selected for use.
06/26/02 : 12:44:26:780 : CCL8405I:
06/26/02 : 12:44:28:441 : CCL8401I: The Following Ciphers are Enabled:
06/26/02 : 12:44:28:442 :       SSL_RSA_WITH_RC4_128_MD5
06/26/02 : 12:44:28:442 :       SSL_RSA_WITH_RC4_128_SHA
06/26/02 : 12:44:28:442 :       SSL_RSA_WITH_DES_CBC_SHA
...
06/26/02 : 12:44:28:457 : CCL6524I: Successfully started handler for the ssl:
protocol.
06/26/02 : 12:44:34:514 : CCL6524I: Successfully started handler for the
systemssl_ssl: protocol.
06/26/02 : 12:44:34:541 : CCL6524I: Successfully started handler for the tcp:
protocol.
```

The message CCL8402I also indicated that the JSSE libraries were selected. This is because we had supplied the name of a keystore file in the CTG.INI file. The CICS TG also outputs a long list of the ciphers that are enabled in the JSSE SSL protocol handler.

3. We checked the state of the TCP/IP sockets on our UNIX System Services TCP/IP stack using the **onetstat** command from OMVS. This showed us the output in Example 8-13 and indicates that the Gateway daemon was indeed listening on port 8052 and 8062 for requests.

Example 8-13 Partial output from the onetstat command

```
$ [SC66] /ctg/scsctg5/logs: onetstat
MVS TCP/IP onetstat CS V1R2      TCPIP Name: TCPIPOE        14:30:48
User Id  Conn     Local Socket          Foreign Socket        State
-------  ----     ------------          --------------        -----
SCSCTG55 00001827 0.0.0.0..8052         0.0.0.0..0            Listen
SCSCTG55 00001825 0.0.0.0..8062         0.0.0.0..0            Listen
```

We could have checked the ports were listening using the ScanPort utility. For further information see “Testing the configuration” on page 52.

4. Next, we checked basic IP connectivity from our Windows 2000 workstation to our z/OS system using the **ping** command from a Windows 2000 prompt (see Example 8-14).

Example 8-14 Output from the ping command on Windows

```
C:\>ping wtsc66oe.itso.ibm.com

Pinging wtsc66oe.itso.ibm.com [9.12.6.29] with 32 bytes of data:

Reply from 9.12.6.29: bytes=32 time=140ms TTL=243
Reply from 9.12.6.29: bytes=32 time=120ms TTL=243
Reply from 9.12.6.29: bytes=32 time=120ms TTL=243
Reply from 9.12.6.29: bytes=32 time=110ms TTL=243

Ping statistics for 9.12.6.29:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 110ms, Maximum =  140ms, Average =  122ms
```

Once we knew everything was working, we then tested from a Windows 2000 workstation. We performed the following steps:

1. We set the CLASSPATH on Windows 2000 to include ctgclient.jar and ctgsamples.jar using the following command:

```
set CLASSPATH=C:\Program Files\IBM\IBM CICS Transaction
Gateway\classes\ctgclient.jar;c:\Program Files\IBM\IBM CICS Transaction
Gateway\classes\ctgsamples.jar;
```

2. We changed the working directory to the CICS TG `bin` directory with the command:

   `cd \Program Files\IBM\IBM CICS Transaction Gateway\bin`

3. We then ran EciB1 using the command in Example 8-15.

Example 8-15 Command to run EciB1 from Windows against SystemSSL

```
java com.ibm.ctg.samples.eci.EciB1 ssl://wtsc66oe.itso.ibm.com 8052
jsseclientss.jks default
```

   – You can see the first parameter after the EciB1 call is the location of the CICS TG server. We specified the `ssl` protocol here.
   – The second parameter on the EciB1 line is the CICS TG port on the z/OS gateway. Our gateway is listening on port 8052 with the System SSL protocol handler, so we specify it here.
   – The third parameter is the SSL classname or keystore. We specified the client keystore we had in the CICS TG bin directory here.
   – The final parameter to EciB1 is the SSL keystore password. We specified the password we had used to protect the keystore here.

   This ran the compiled version of EciB1 inside ctgsamples.jar and connected to the CICS TG on our z/OS system, using the SSL support provided by JSSE on Windows and SystemSSL on z/OS. The jsseclientss.jks keystore specified when we ran the command was in the working directory.

4. We entered `1` at the prompt to select our CICS server. The result of our test is given in Example 8-16. As shown, EC01 returned the time and date on our CICS server.

Example 8-16 Output from EciB1

```
CICS Transaction Gateway Basic ECI Sample 1
-------------------------------------------
Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway Url]
                                          [Gateway Port Number]
                                          [SSL Classname]
                                          [SSL Password]

To enable client tracing, run the sample with the following Java option:
 -Dgateway.T.trace=on

The address of the Gateway has been set to ssl://wtsc66oe.itso.ibm.com
Port:8052
```

```
CICS Servers Defined:

        1. SCSCPJA1 -


Choose Server to connect to, or q to quit:
1
Program EC01 returned with data:-

        Hex: 32362f30362f30322031363a33303a30310
        ASCII text: 26/06/02 16:30:01
```

5. We then tested the JSSE SSL protocol handler. We ran EciB1 using the command in Example 8-17.

Example 8-17 Command to run EciB1 from Windows against SSL

```
java com.ibm.ctg.samples.eci.EciB1 ssl://wtsc66oe.itso.ibm.com 8062
jsseclientss.jks default
```

   – The CLASSPATH is identical to the System SSL test. This is because from the client we use the JSSE libraries to test both SSL protocol handlers.

   – The parameters on the EciB1 call are the same as those for the System SSL test with the exception of the port number. We specified port 8062, since this was where the JSSE SSL protocol handler was listening. The other parameters are the same because, from the client perspective, we are using the same protocol to communicate with the CICS TG. We use the same keystore because it contains the self-signed certificate we added in “Client keystore for SystemSSL and JSSE” on page 196.

   This ran the compiled version of EciB1 inside ctgsamples.jar and connected to the CICS TG on our z/OS system, using the SSL support provided by JSSE on Windows and JSSE on z/OS. The only difference from the previous command is that we specified port 8062 where the SSL protocol handler was listening.

6. We entered 1 at the prompt to select our CICS server. The successful output of EciB1 was again identical to that shown in Example 8-16 on page 206, apart from the date returned by EC01.

8.4 Problem determination

In this section, we document information we learned while configuring this scenario, and further information on problem determination and tracing.

8.4.1 Tips and utilities

We found the keytool utility useful for verifying that we had specified the correct password for a keystore and that a keystore was intact. The following **`keytool`** command lists the contents of the keystore jsseclientss.jks using the password `default`:

```
keytool -list -keystore jsseclientss.jks -storepass default
```

If the password was incorrect, the following error is output:

```
keytool error: java.io.IOException: Keystore was tampered with, or password
was incorrect
```

If the keystore is corrupt, the following error is output:

```
keytool error: java.io.IOException: Invalid keystore format
```

In the following sections, we detail common errors we came across when setting up JSSE and testing SSL.

CCL6525E Unable to start handler for the ssl: protocol

This error message can appear in the CICS TG log for a number of reasons. The CICS TG will also give the exception message that occurred in the underlying library. This can be used to determine what went wrong.

- If the keystore password is incorrect, the message will be `[java.io.IOException: Keystore was tampered with, or password was incorrect]`. An incorrect password is the most common cause of this error. Verify the keystore password using the **`keytool`** command.
- The message `[java.io.IOException: Algorithm IbmX509 not available]` indicates that the JSSE libraries are not installed correctly. This is caused by not having the correct versions of ibmjsse.jar, ibmjcefw.jar, ibmjceprovider.jar in the JDK's `ext` directory.
- If the message `CCL8403I SSLight libraries selected for use` is output earlier in the CICS TG log, and a JSSE keystore is being used, the JSSE libraries are not installed in the JDK used by the CICS TG. The CCL6525E message will be followed by `[java.io.IOException: Keystore was tampered with, or password was incorrect]`.

- If the keystore is not in the location referenced by CTG.INI, the message will state `java.io.FileNotFoundException`, will give the file name specified for the keystore, and will state (`The system cannot find the file specified`).
- If the port number the CICS TG is trying to listen on is already in use by another system, the message will state [`java.net.BindException: Address in use: bind`]. The **onetstat** command can be used to see if another system is using the desired port.

Unable to start handler for the system_ssl: protocol

The error message `CCL6525E: Unable to start handler for the system_ssl: protocol` can appear in the CICS TG log for a number of reasons. The CICS TG will also give the exception message that occurred in the underlying library. This can be used to determine what went wrong.

- If the key database password is incorrect, the message will be [`java.io.IOException: rc=4`]. Check that the key database password specified is correct by trying to open it using **gskkyman**.
- If the message reads [`java.io.IOException: rc=2`], we found one of two errors can cause this:
  - The key database (.kdb and .rdb files) is not in the location specified in CTG.INI, so they cannot be found by the CICS TG.
  - The key database file does not have the correct permissions to allow the CICS TG to read it. Check that the file permissions are correct with the command **ls -l systemssl.kdb**.
- If the port number the CICS TG is trying to listen on is already in use by another system, the message will state [`java.io.IOException: rc=-1`]. The **onetstat** command can be used to see if another system is using the desired port.

CCL6651E: Unable to connect to the Gateway

This error message can appear on the Java client application when something goes wrong connecting to the CICS TG using SSL.

- If the message [`java.io.IOException: Invalid keystore format`] appears and the application is being run on z/OS, check that binary mode was used to upload the client keystore to the z/OS system. Trying to use a keystore that was uploaded using ASCII mode will cause this error. Verify that the keystore is intact using the keytool utility.
- The message [`java.io.IOException: Keystore was tampered with, or password was incorrect`] again strongly suggests that the keystore password supplied by the client application is incorrect. Verify the keystore password using the **keytool** command.

- If the keystore cannot be found by the Java application, the message will state `java.io.FileNotFoundException`, will give the file name specified for the keystore, and will state (`The system cannot find the file specified`). Check that the filename specified for the keystore is correct.

CCL6668E: Initial handshake flow failed

This error message appears on the Java client application when the SSL handshake with the CICS TG fails.

- If the message is `[javax.net.ssl.SSLHandshakeException: unknown certificate]`, then the signer certificate of the key used in the CICS TG server is not present in the keystore used by the client Java application. This happens if you try to use the SSL-only keystore jsseclientsslonly.jks to connect to the System SSL protocol handler, or if the self-signed CICS TG certificate is not imported into the client keystore. iKeyman or keytool can be used to view the certificates present in a keystore.
- If the message states `[ERROR_CONNECTION_FAILED]`, check that the CICS TG protocol has been specified as `ssl://` and *not* `tcp://`. Using the TCP protocol to connect to an SSL protocol handler will generate this error.
- If the message contains `[javax.net.ssl.SSLProtocolException: end of file]`, check that the application has specified an SSL protocol handler and *not* a TCP protocol handler. Trying to connect to a TCP protocol handler using `ssl://` will also result in this error. Also, check that the Gateway daemon certificate has not expired. Using an expired System SSL certificate will generate this error on the client application.

iKeyman error message when loading a keystore

When loading a keystore into iKeyman the window in Figure 8-11 might appear. This can be caused by specifying the wrong password at the password prompt. The same window is caused when the keystore is corrupt, if it was transferred using FTP in ASCII mode for example.

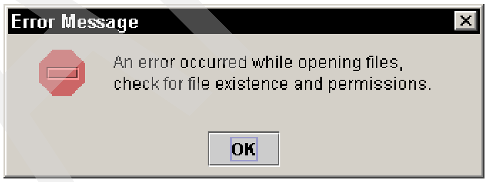


Figure 8-11 iKeyman loading error message

Testing JSSE under z/OS

We found running the following test from z/OS UNIX System Services useful to quickly verify that the JSSE SSL protocol handler was working correctly. We performed the following steps:

1. From a z/OS UNIX System Services shell, we changed to the /ctg/scsctg5 directory where our server keystore was.
2. We used the **keytool** command in Example 8-8 on page 199 to invoke the Key Management Tool to export the self-signed certificate from the server keystore in the file jssesslss.jks into a file called server.der.
3. We used the **keytool** command in Example 8-18 to invoke the Key Management Tool to import the self-signed certificate from the file server.der into the keystore jsseclientsslonly.jks, identified by the alias `ITSO`. Because this keystore did not exist, **keytool** created it. The keystore password was set to `default`.

Example 8-18 Creating a client keystore with the self-signed certificate

```
$ [SC66] /ctg/scsctg5: keytool -import -alias ITSO -file server.der
-keystore jsseclientsslonly.jks -storepass default
Certificate was added to keystore
```

This added our self-signed server certificate into the keystore as a trusted CA.

4. We used the commands in Example 8-19 to set the CLASSPATH and run EciB1 against the JSSE SSL protocol handler.

Example 8-19 Commands to test JSSE SSL using the SSL-only keystore

```
export CLASSPATH=/usr/lpp/ctg500/ctg/classes/ctgclient.jar
export CLASSPATH=$CLASSPATH:/usr/lpp/ctg500/ctg/classes/ctgsamples.jar
java com.ibm.ctg.samples.eci.EciB1 ssl://wtsc66oe.itso.ibm.com 8062
/ctg/scsctg5/jsseclientsslonly.jks default
```

The commands ran the compiled version of EciB1 inside ctgsamples.jar and connected to the CICS TG on our z/OS system using the SSL support provided by the JSSE library on both client and CICS TG. We entered `1` at the prompt to select our CICS server. The results of our test was identical to that shown in Example 8-16 on page 206, except the time and date on our CICS server was different.

8.4.2 Tracing

For information on tracing the CICS TG on z/OS, refer to the tracing section of Chapter 7, “TCP connections to the Gateway daemon on z/OS” on page 133.

9

TCP connections to the Gateway daemon on Linux

In this chapter, we show you how we implemented a TCP connection from a remote Java client application to our CICS Transaction Gateway (CICS TG) running on Linux for S/390.

This chapter covers the following topics:

- Preparation
- Configuration
- Problem determination

For details on how we configured the TCP/IP connection from the CICS TG on Linux to our CICS TS region on z/OS, refer to Chapter 5, “TCP/IP connections to CICS” on page 81.

9.1 Preparation

The following sections details the software levels we used in our sample configuration (Figure 9-1), and some instructions on how to install the CICS TG for Linux in quick and proper fashion, to begin your configuration.

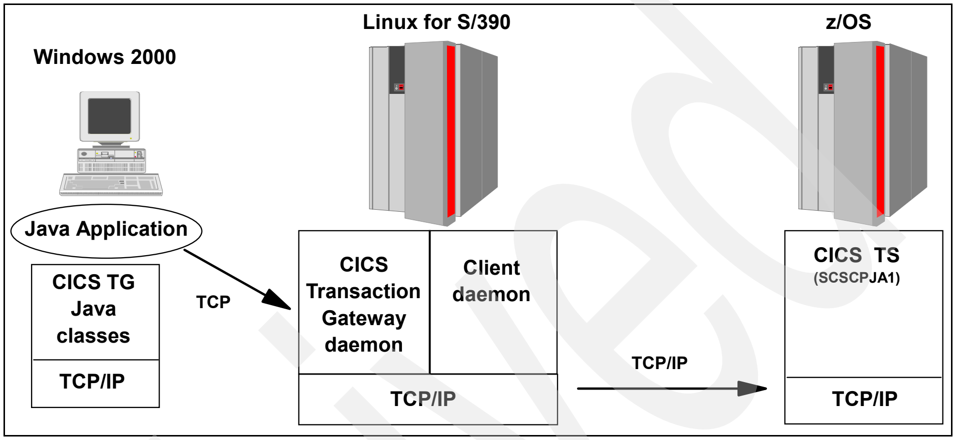


Figure 9-1 Software components: CICS TG on Linux

9.1.1 Software checklist

We used the levels of software described in Table 9-1.

Table 9-1 Software checklist: CICS TG on Linux

Client workstation	Server system	z/OS
Windows 2000 Service Pack 2	SuSE Linux Enterprise Server 7 for S/390 and zSeries®	z/OS V1.2
CICS Transaction Gateway for Windows V5.0	CICS Transaction Gateway for Linux V5.0	CICS Transaction Server V2.2
IBM Java Development Kit V1.3.0	IBM Java Development Kit V1.3.1	IBM Java Development Kit V1.3.1
		Communications Server V2.8 (includes VTAM and TCP/IP)

9.1.2 Definitions checklist

The definitions we used in this scenario are summarized in Table 9-2. Before you configure the products, we recommend that you acquire definitions for each parameter listed.

Table 9-2 Definitions checklist: CICS TG on Linux

CICS Transaction Gateway	CICS Transaction Server	Example
hostname		vmlinux1.itso.ibm.com
CICS TG network protocol		tcp:
port		2006
CICS Server name	APPLID	SCSCPJA1
CICS Server hostname		wtsc66oe.itso.ibm.com
CICS Server TCP/IP port	TCPIPSERVICE PORT	8018

Note that we also had to deploy the sample CICS COBOL application EC01, which is provided in the server directory in the CICS TG samples directory.

We also had to configure a DFHCNV data conversion template in CICS for use by this program. Refer to Appendix A, "DFHCNV and CICS data conversion" on page 329 for more details.

9.1.3 Installation of the CICS TG

The following section details how we installed the CICS TG on our Linux system.

Enabling the FTP service on Linux

We had to enable the FTP service on Linux. We did this as follows:

1. We edited the Linux internet *super-server* (inetd) configuration file /etc/inetd.conf and uncommented the line defining the FTP service, as shown in Example 9-1.

Example 9-1 Enabling the FTP service in /etc/inetd.conf

```
# These are standard services.
#
# ftp   stream  tcp   nowait  root    /usr/sbin/tcpd  wu.ftpd -a
ftp     stream  tcp   nowait  root    /usr/sbin/tcpd  proftpd
# ftp   stream  tcp   nowait  root    /usr/sbin/tcpdin.ftpd
```

2. We checked that the user ID we were going to use to log into the FTP service was not listed in the file /etc/ftpusers. This file defines users who are not allowed to log into the FTP service.
3. We used the command **/etc/init.d/inetd restart** to restart the inetd daemon, the output of which is shown in Example 9-2.

Example 9-2 Restarting the inetd daemon

```
root@vmlinux1:/etc > /etc/init.d/inetd restart
Shutting down inetd                                                    done
Starting inetd                                                         done
```

> **Note:** Here we enabled the proftpd FTP server daemon. We could have enabled one of the other FTP server daemons in the inetd configuration file, for example in.ftpd, but we preferred proftpd because it is more configurable.

The Red Hat Package Manager

Before showing how we installed the CICS TG, we should first explain how it is installed onto Linux using the Red Hat Package Manager (RPM).

The Red Hat Package Manager is a powerful package manager, present on many Linux distributions, which can be used to install, query, update, and uninstall individual software packages. A package consists of an archive of files and package information, including name, version and description. The Red Hat Package Manager maintains a database of software packages installed onto the system, which users can interact with using the **rpm** command.

When the CICS TG install process is run, the **rpm** command is used to install the CICS TG onto the system. Once the CICS TG is installed, **rpm** can be used to display, among other things, details of the CICS TG and where it is installed. It is also used to uninstall the CICS TG.

Transferring the CICS TG files to Linux

The CICS TG is supplied as an rpm file called ctg-5.0.0-1l.s390.rpm, which must be uploaded onto the Linux file system.

We transferred it directly to the /tmp directory on our Linux system. Example 9-3 shows the transcript of our FTP session. Note that we used binary mode for the transfer, as we did not want FTP to perform data conversion on the tar archive.

Example 9-3 Example FTP upload session

```
C:\temp>ftp vmlinux1.itso.ibm.com
Connected to vmlinux1.itso.ibm.com.
220 ProFTPD 1.2.2rc2 Server (powered by SuSE Linux) [vmlinux1.itso.ibm.com]
```

```
User (vmlinux1.itso.ibm.com:(none)): cicsrs3
331 Password required for cicsrs3.
Password:
230 User cicsrs3 logged in.
ftp> cd /tmp
250 CWD command successful.
ftp> bin
200 Type set to I.
ftp> put ctg-5.0.0-1.s390.rpm
200 PORT command successful.
150 Opening BINARY mode data connection for  ctg-5.0.0-1.s390.rpm.
226 Transfer complete.
```

Once we had uploaded the CICS TG install RPM, we performed the following steps.

Installing the CICS TG

We installed the CICS TG RPM using the command **`rpm -Uvh ctg-5.0.0-1.s390.rpm`**, which made the Red Hat Package Manager program install the CICS TG into the /opt/ctg/ directory. We then ran the CICS TG install script ctginstall, which was created when the CICS TG was installed. The transcript is shown in Example 9-4.

Example 9-4 Installing the CICS TG

```
root@vmlinux1:/ > cd /tmp
root@vmlinux1:/tmp > rpm -Uvh ctg-5.0.0-1.s390.rpm
ctg                         ##################################################
Run the command 'ctginstall' to complete the installation.
root@vmlinux1:/tmp >ctginstall
```

This displayed the CICS TG install prompt asking us to enter `1` to view the license agreement, as shown in Example 9-5.

Example 9-5 CICS TG install script license prompt

```
To install CICS Transaction Gateway you must accept the following license
agreement.
If the license file does not display
correctly, try restarting the
installation using the command:
'/opt/ctg/bin/ctgsetup 40'.
Enter 1 to view the license, or 2 to quit.
```

We viewed the agreement by entering `1`. We entered `1` repeatedly until we had viewed the entire license agreement, as shown in Example 9-6 on page 218.

Example 9-6 CICS TG license agreement prompt

```
Specified Operating Environment

The Program's specifications and specified operating environment information
may be found in documentation accompanying the Program, if available, such as a
read-me file, or other information published by IBM, such as an announcement
letter.

U.S. Government Users Restricted Rights

U.S. Government Users Restricted Rights - Use, duplication, or disclosure
restricted by the GSA ADP Schedule Contract with the IBM Corporation.

D/N: L-TMAN-57QH8C
P/N: L-TMAN-57QH8C
<<< End of the License File <<<
________________________________________________________________________________
Enter 1 to page down, 2 to page up, 3 to accept or 4 to decline.
```

We entered 3 and pressed Enter to accept the license agreement. The install program then performed some customization steps necessary to run the CICS TG. When finished, the install program output the completion messages shown in Example 9-7.

Example 9-7 Installing the CICS TG, completion messages

```
3
You have accepted the license agreement.
Wait while the installation continues.
The installation is complete.
License files can be viewed using the command: '/opt/ctg/bin/ctgbrowse'
To view the PDF documentation, use a PDF reader such as 'xpdf' or 'Ghostview'.
If you are installing for the first time on this machine, you may need to
change the maximum sizes for IPC messages and message queues. Refer to the CTG
Administration book.
Now run command 'ctgcfg' to configure the CICS Transaction Gateway.
```

Verifying the install

We verified that the CICS TG was installed to the system using the following Red Hat Package Manager (RPM) command to query the system software package database:

```
rpm -qi ctg
```

This output information about the CICS TG package, including the date it was installed, is shown in Example 9-8 on page 219.

Example 9-8 Using the rpm command to query the installed CICS TG

```
root@vmlinux1:/tmp > rpm -qi ctg
Name        : ctg                          Relocations: (not relocateable)
Version     : 5.0.0                             Vendor: IBM Corporation.
Release     : 1                             Build Date: Fri 21 Jun 2002
11:30:17 AM EST
Install date: Mon 01 Jul 2002 05:32:47 PM EDT      Build Host:
winlnx01.hursley.ibm.com
Group       : Application/Communications    Source RPM: ctg-5.0.0-1.src.rpm
Size        : 30170845                         License: IBM OCO - see other
documentation for licence terms.
URL         : http://www.ibm.com/software/ts/cics
Summary     : IBM CICS Transaction Gateway for Linux.
Description :
This is the IBM CICS Transaction Gateway for Linux.
It provides connectivity to CICS servers on various platforms from Java, C
and C++ programs.  It provides APIs for ECI, EPI and ESI (SNA only) access.
```

The result of our CICS TG install was the directory structure shown in Figure 9-2.

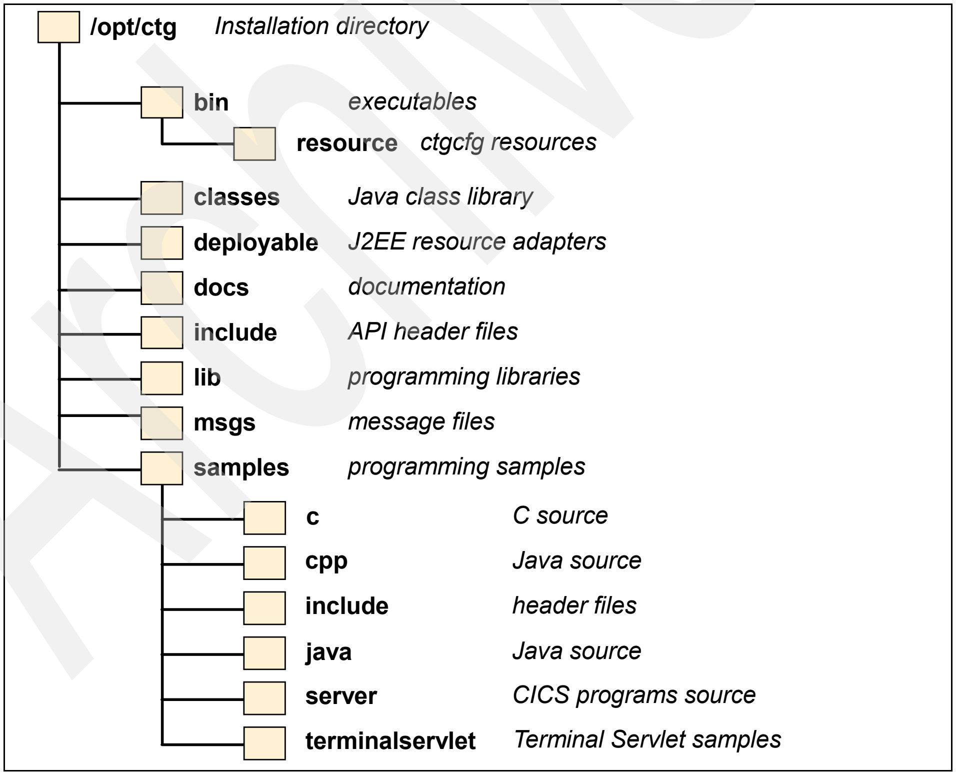


Figure 9-2 CICS TG on Linux directory structure

The directory /opt/ctg/bin contains the following files, all of which can be browsed using the command `more`:

CTGSAMP.INI Sample CTG.INI configuration file.

ctgcfg Shell script to start CICS TG X-Windows graphical Configuration Tool.

ctgstart Shell script to start the Gateway daemon.

ctgikey Shell script to start the SSLight tool iKeyman.

The other files in the directory are executables and cannot be viewed.

The /opt/ctg/classes directory contains the following Java class libraries of note:

ctgclient.jar CICS TG Java class library.

ctgserver.jar CICS TG classes for use by Gateway daemon.

ctgsamples.jar CICS TG Java samples library.

The contents of these can be listed with the command `jar -tvf <file>`. The ctgsamples.jar file contains compiled versions of all the non-J2EE samples. We found this useful for testing because we could execute the samples easily using just this JAR file and the CICS TG Java class library JAR contained in ctgclient.jar.

9.2 Configuration

In this section, we detail how we configured the software components on Windows 2000, Linux, and z/OS for this scenario.

To listen for requests, we needed to enable the TCP protocol handler. We did this using the Configuration Tool. To run the Configuration Tool on Linux, we ran an X-Windows server on our Windows 2000 workstation, and specified the address of the X-Windows server in the `DISPLAY` environment variable on Linux. The IP address of our Windows 2000 machine was 9.1.39.10 in this case. We then ran the Configuration Tool, as shown in Example 9-9.

Example 9-9 Setting Linux to use the Windows 2000 X-Server for the Configuration Tool

```
root@vmlinux1:/opt/ctg/bin > export DISPLAY=9.1.39.10:0
root@vmlinux1:/opt/ctg/bin > ctgcfg
```

> **Tip:** The Configuration Tool can also be used on Windows to create the configuration file. This is detailed in “Using Windows to configure the CICS TG on Linux” on page 222.

We clicked **No** on the TaskGuide? window that appeared.

From the Configuration Tool main menu, we clicked the **TCP** protocol handler in the Java gateway tree, and then ticked the **Enable protocol handler** checkbox (Figure 9-3).

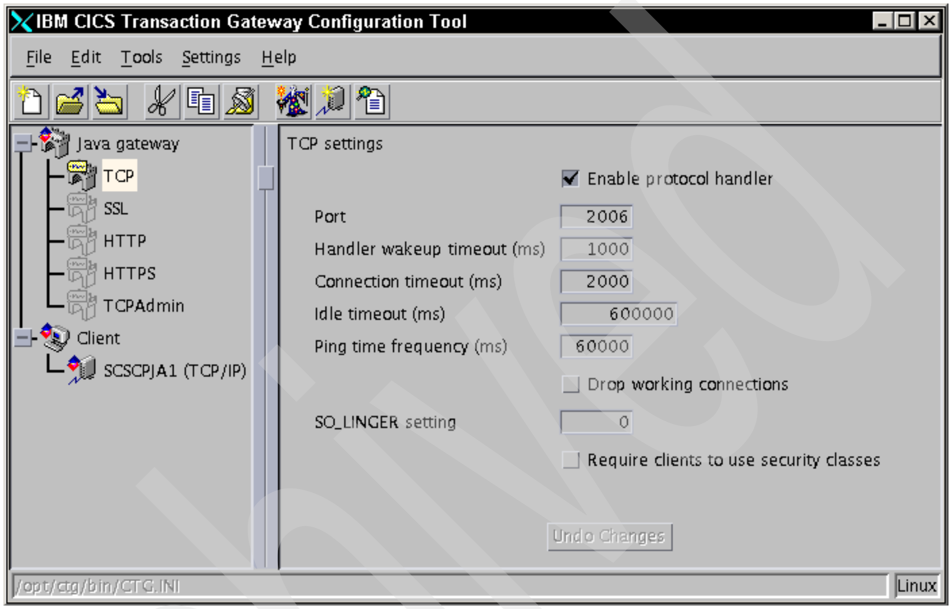


Figure 9-3 CICS TG Linux, configuring the TCP handler

This causes the CICS TG TCP protocol handler to start up when the Gateway daemon starts, and listen for TCP requests on port 2006. It creates the lines shown in Example 9-10 in the CTG.INI file.

Example 9-10 TCP protocol handler lines in CTG.INI

```
protocol@tcp.handler=com.ibm.ctg.server.TCPHandler

protocol@tcp.parameters=connecttimeout=2000;idletimeout=600000;pingfrequency=60
000;port=2006;solinger=0;sotimeout=1000;
```

We also needed to configure a connection from the Client daemon to our CICS region SCSCPJA1 on z/OS. For this, we re-used the TCP/IP connection described in Chapter 5, “TCP/IP connections to CICS” on page 81. See Figure 9-4 on page 222 for the definitions used. This creates the lines shown in Example 9-11 on page 222 in the CTG.INI file.

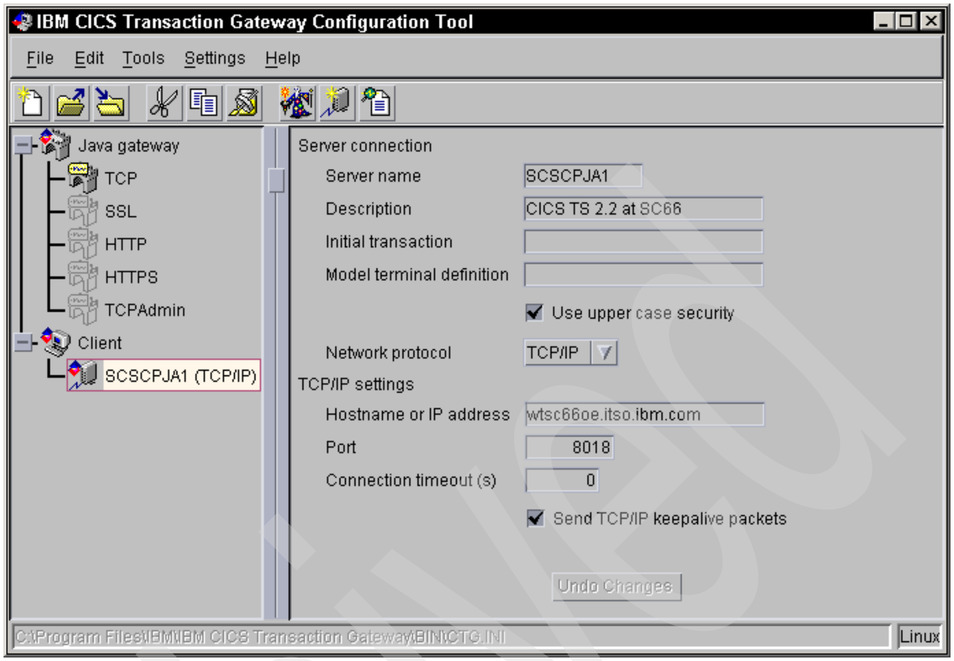


Figure 9-4 CICS TG Linux, configuring the server definition

Example 9-11 Server definition in CTG.INI

```
SECTION SERVER = SCSCPJA1
    DESCRIPTION=CICSTS 2.2 AT SC66
    UPPERCASESECURITY=Y
    PROTOCOL=TCPIP
    NETNAME=wtsc66oe.itso.ibm.com
    PORT=8018
    CONNECTTIMEOUT=0
    TCPKEEPALIVE=Y
ENDSECTION
```

Using Windows to configure the CICS TG on Linux

When configuring the CICS TG from Linux, we found the Configuration Tool was very slow due to poor network performance. We instead chose to run the Configuration Tool on Windows to define our configuration for the CICS TG on Linux. We did this as follows:

1. We started the Configuration Tool on our Windows 2000 machine with a special parameter that caused it to configure for Linux, using the following command:

   ```
   ctgcfg -PLAT L390
   ```

2. We defined our configuration using the Configuration Tool on Windows and saved the CTG.INI file into the c:\ctg\bin directory. This can be seen in Figure 9-4 on page 222. Note that the Configuration Tool looks almost identical to the Linux version, except the location of the CTG.INI file is shown as being in the Windows directory C:\Program Files\IBM\IBM CICS Transaction Gateway\bin rather than the Linux directory /opt/ctg/bin.
3. We uploaded the CTG.INI file from our Windows 2000 machine onto our Linux system into the /tmp directory. Example 9-12 shows the transcript of our FTP session. Note that we used ASCII mode for the transfer, since we wanted FTP to perform data conversion on the ASCII file.

Example 9-12 Uploading the CTG.INI file

```
C:\>cd \ctg\bin
C:\ctg\bin>ftp vmlinux1.itso.ibm.com
Connected to vmlinux1.itso.ibm.com.
220 ProFTPD 1.2.2rc2 Server (powered by SuSE Linux) [vmlinux1.itso.ibm.com]
User (vmlinux1.itso.ibm.com:(none)): itso
331 Password required for itso.
Password:
230 User itso logged in.
ftp> ascii
200 Type set to A.
ftp> cd /tmp
250 CWD command successful.
ftp> put CTG.INI
200 PORT command successful.
150 Opening ASCII mode data connection for CTG.INI.
226 Transfer complete.
ftp: 1067 bytes sent in 0.00Seconds 1067000.00Kbytes/sec.
```

4. We moved the CTG.INI file on our Linux system from the /tmp directory to the /opt/ctg/bin directory using the following command at the Linux command prompt:

```
mv /tmp/CTG.INI /opt/ctg/bin
```

Note: We found that CICS TG configuration files are generally portable between different UNIX platforms and between Windows and UNIX. However, the name of the Client daemon TCP/IP protocol driver is different on Windows and on UNIX. Using a Windows configuration file with a TCP/IP server connection defined on a UNIX platform results in error messages in the Client daemon log:

```
[3251] CCL:CCL3247 Error loading DLL '/opt/ctg/bin/CCLWNTIP.a'
(/opt/ctg/bin/CCLWNTIP.a: cannot open shared object file: No such file or
directory, 0)
[1149] POP:CCL3229E Cannot load protocol driver 'CCLWNTIP'
```

Running the Configuration Tool with the `-plat` option, specifying the platform to configure for, avoids such problems.

9.3 Testing the configuration

To test our configuration we used the CICS TG sample Java application EciB1. This is installed in the CICS TG samples directory, inside the java subdirectory. The sample's Java class is defined to be in the Java package `com.ibm.ctg.samples.eci`, so the sample source file is stored under the following directory structure:

```
com/ibm/ctg/samples/eci
```

The full path to the EciB1 sample on our Linux system is:

```
/opt/ctg/samples/java/com/ibm/ctg/samples/eci/EciB1.java
```

and on our Windows system it is:

```
C:\Program Files\IBM\IBM CICS Transaction
Gateway\samples\java\com\ibm\ctg\samples\eci\EciB1.java
```

The sample is also provided in a compiled form inside ctgsamples.jar.

We used EciB1 from our Linux system to call the CICS program EC01. The EciB1 application flows an ECI request to a connected CICS region through a specified CICS TG (Figure 9-5 on page 225) and invokes the transaction EC01. The CICS TG URL is specified as an input parameter, and the CICS region is entered at the interactive prompt. We then used EciB1 from a Window 2000 workstation to call the same CICS program (Figure 9-6 on page 225).

After we installed and configured the software components as illustrated in Figure 9-6, we tested our configurations as follows:

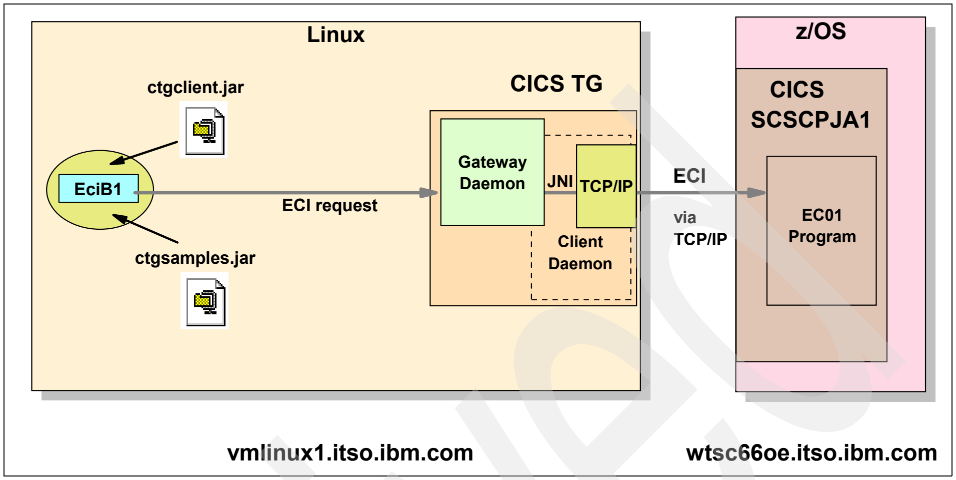


Figure 9-5 CICS TG Linux TCP, software configuration for Linux testing

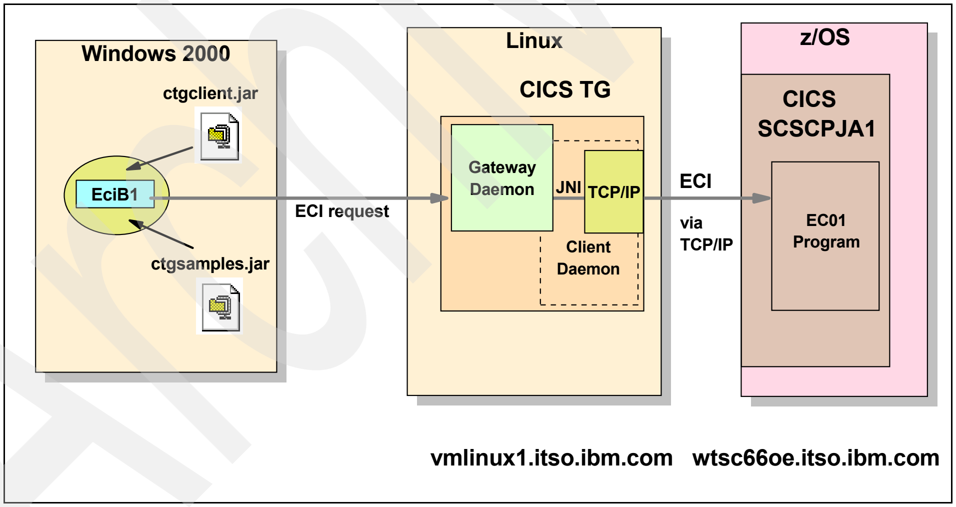


Figure 9-6 CICS TG Linux, TCP software configuration for Windows testing

1. We started the Gateway daemon on Linux using the **ctgstart** command, from the CICS TG bin directory, as shown in Example 9-13 on page 226. This caused the CICS TG to start listening on port 2006 for TCP requests.

2. We checked the CICS TG log (displayed when the CICS TG is started) for the message CCL6524I indicating the TCP protocol handler had started (see Example 9-13).

Example 9-13 The CICS TG message log

```
root@vmlinux1:/ > cd /opt/ctg/bin
root@vmlinux1:/opt/ctg/bin > ctgstart
07/01/02 : 14:58:44:819 : CICS Transaction Gateway, Version 5.0.0 Pre-release,
5724-D12. Build Level c000-20020621.
07/01/02 : 14:58:44:830 : (C) Copyright IBM Corporation 1999, 2002. All rights
reserved.
07/01/02 : 14:58:44:953 : CCL8400I: Using ini file /opt/ctg/bin/CTG.INI.
07/01/02 : 14:58:44:956 : CCL6577I: Java version is 1.3.1.
07/01/02 : 14:58:44:959 : CCL6502I: Initial ConnectionManagers = 1, Maximum
ConnectionManagers = 100,
07/01/02 : 14:58:45:236 : CCL6502I: Initial Workers = 1, Maximum Workers = 100,
tcp: Port = 2006
07/01/02 : 14:58:45:237 : CCL6574I: Connection logging has been disabled.
07/01/02 : 14:58:45:292 : CCL6505I: Successfully created the initial
ConnectionManager and Worker threads.
07/01/02 : 14:58:45:352 : CCL6524I: Successfully started handler for the tcp:
protocol.
```

3. We checked the state of the TCP/IP sockets on our Linux machine using the **`netstat -l -n`** command from a Linux command prompt. This showed us that the CICS TG was indeed listening on port 2006 for TCP requests (see Example 9-14).

Example 9-14 Output from the netstat command

```
tcp        0      0 0.0.0.0:32768           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:79              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:2006            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
```

4. Next we checked basic IP connectivity from our Windows 2000 workstation to our Linux system using the **`ping`** command from a Windows 2000 prompt (see Example 9-15), and from our Linux system to our z/OS system using the **`ping`** command from a Linux prompt (see Example 9-16 on page 227).

Example 9-15 Output from the ping command on Windows

```
C:\>ping vmlinux1.itso.ibm.com

Pinging vmlinux1.itso.ibm.com [9.12.6.98] with 32 bytes of data:
```

```
Reply from 9.12.6.98: bytes=32 time=140ms TTL=243
Reply from 9.12.6.98: bytes=32 time=120ms TTL=243
Reply from 9.12.6.98: bytes=32 time=120ms TTL=243
Reply from 9.12.6.98: bytes=32 time=110ms TTL=243

Ping statistics for 9.12.6.98:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 110ms, Maximum =  140ms, Average =  122ms
```

Example 9-16 Output from the ping command on Linux

```
root@vmlinux1:~ > ping wtsc66oe.itso.ibm.com
PING wtsc66oe.itso.ibm.com (9.12.6.29): 56 data bytes
64 bytes from 9.12.6.29: icmp_seq=0 ttl=64 time=1.715 ms
64 bytes from 9.12.6.29: icmp_seq=1 ttl=64 time=1.106 ms
64 bytes from 9.12.6.29: icmp_seq=2 ttl=64 time=1.519 ms
64 bytes from 9.12.6.29: icmp_seq=3 ttl=64 time=1.276 ms
--- wtsc66oe.itso.ibm.com ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 1.106/1.404/1.715 ms
```

5. We then checked that CICS was listening on the TCP/IP port configured in the Client daemon server connection using the ScanPort utility. See Chapter 5.4, “Problem determination” on page 90 for more details.
6. We knew that everything was working, so we set the CLASSPATH on Linux to include ctgclient.jar and ctgsamples.jar using the following command:

   ```
   export CLASSPATH=/opt/ctg/classes/ctgclient.jar:/opt/ctg/
   classes/ctgsamples.jar
   ```
7. We then ran the EciB1 Java application on our Linux system using the command in Example 9-17. This ran the compiled version of EciB1 inside ctgsamples.jar and connected to the CICS TG on our Linux system. We entered 1 at the prompt to select our CICS server. As shown, EC01 returned the time and date on our CICS server.

Example 9-17 Output from EciB1

```
root@vmlinux1:/ > java com.ibm.ctg.samples.eci.EciB1
tcp://vmlinux1.itso.ibm.com

CICS Transaction Gateway Basic ECI Sample 1
-------------------------------------------
Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway Url]
                                          [Gateway Port Number]
                                          [SSL Classname]
                                          [SSL Password]
```

```
To enable client tracing, run the sample with the following Java option:
 -Dgateway.T.trace=on

The address of the Gateway has been set to tcp://vmlinux1.itso.ibm.com
Port:2006


CICS Servers Defined:

         1. SCSCPJA1 -SCSCPJA1 TCPIP

Choose Server to connect to, or q to quit:
1

Program EC01 returned with data:-

        Hex: 30342f30362f30322031383a31313a30380
        ASCII text: 04/06/02 18:11:08
```

Note: We used the TCP protocol to connect to the Gateway daemon on our Linux system. If we did not specify the address of the CICS TG to connect to, EciB1 would have used local mode and would still run the CICS program successfully. However, local mode does not use the Gateway daemon and would therefore not have tested the Gateway daemon we had running on Linux.

8. Once connectivity from Linux was working, we then tested from a Windows 2000 workstation. We set the CLASSPATH on Windows 2000 to include ctgclient.jar and ctgsamples.jar using the following command:

```
set CLASSPATH=C:\Program Files\IBM\IBM CICS Transaction
Gateway\classes\ctgclient.jar;c:\Program Files\IBM\IBM CICS Transaction
Gateway\classes\ctgsamples.jar;
```

9. We then ran the EciB1 Java application on our Windows 2000 workstation using the following command:

```
java com.ibm.ctg.samples.eci.EciB1 tcp://vmlinux1.itso.ibm.com
```

This again ran the compiled version of EciB1 inside ctgsamples.jar and connected to the CICS TG on our Linux system. We entered 1 at the prompt to select our CICS server. The successful output of EciB1 was almost identical to that shown in Example 9-17 on page 227, apart from the date returned by EC01.

Compiling and using EciB1

In the above test we used the compiled version of EciB1 inside the CICS TG samples JAR. In some cases we needed to alter EciB1 and run the modified version. To compile the EciB1 source file EciB1.java stored in the samples directory and then run it, we followed these steps from a Windows command prompt:

1. We set the CLASSPATH to include the current directory and the CICS TG Java class library:

   ```
   set CLASSPATH=.;C:\Program Files\IBM\IBM CICS Transaction
   Gateway\classes\ctgclient.jar
   ```

2. We changed directory into the java subdirectory in the CICS TG samples directory:

   ```
   cd c:\Program Files\IBM\IBM CICS Transaction Gateway\samples\java
   ```

3. We compiled the source file EciB1.java in the samples directory, which was stored under a directory hierarchy reflecting its Java package. We used the following command:

   ```
   javac com\ibm\ctg\samples\eci\EciB1.java
   ```

4. We ran the copy of EciB1 we had just compiled in the samples directory:

   ```
   java com.ibm.ctg.samples.eci.EciB1 tcp://vmlinux1.itso.ibm.com
   ```

The output of the session can be seen in Example 9-18.

Example 9-18 Command session for compiling and running EciB1

```
C:\>set CLASSPATH=.;c:\Program Files\IBM\IBM CICS Transaction
Gateway\classes\ctgclient.jar
C:\>cd c:\Program Files\IBM\IBM CICS Transaction Gateway\samples\java
C:\ctg\samples\java>javac com\ibm\ctg\samples\eci\EciB1.java
C:\ctg\samples\java>java com.ibm.ctg.samples.eci.EciB1
tcp://vmlinux1.itso.ibm.com
```

Note: We set the CLASSPATH to include the current directory (the **.** character) because, when we executed the command to run EciB1, the Java interpreter looked for the file com\ibm\ctg\samples\eci\EciB1.class relative to any directories in the classpath. Because the current directory was c:\Program Files\IBM\IBM CICS Transaction Gateway\samples\java when we ran the command, the Java interpreter was able to find this file and successfully execute our modified version of EciB1.

9.4 Problem determination

In this section, we document information we learned while configuring this scenario, and further information on problem determination and tracing.

9.4.1 Tips and utilities

In this section, you will find useful commands and utilities for debugging problems with your configuration. You will also find additional information about topics discussed in this chapter.

Saving the CICS TG messages

If you wish to save the whole message log, issue the following commands to redirect the output to a log file:

```
cd /opt/ctg/bin
ctgstart >ctg.log 2>&1
```

Note that you will now need to use another command prompt to view the log file, as follows:

```
cat /opt/ctg/bin/ctg.log
```

Alternatively you can save the message log while having it output to the screen using the tee utility as follows:

```
cd /opt/ctg/bin
ctgstart 2>&1 |tee ctg.log
```

Error messages

The Client daemon writes error messages to the file CICSCLI.LOG, located in the /var/cicscli/ directory. This file is created or written to only in error situations and can be viewed using any text editor. It is particularly useful as a first stop for determining if problems have occurred with a TCP connection to a CICS server.

9.4.2 Tracing

There are three ways to trace the CICS Transaction Gateway:

- Gateway daemon tracing, with various levels of tracing available.
- Gateway daemon Java Native Interface (JNI) trace.
- Java client tracing, which can be used to trace the Java method calls made using the CICS TG Java classes in the Java client.

Gateway daemon trace

Gateway daemon trace can be controlled at startup of the Gateway with the following startup options:

-trace This enables extra tracing messages.

-x This enables full debug tracing.

-tfile=<*pathname*> This option can be used in conjunction with both the -trace and -x options, and overrides the default location of ctg.trc file in the bin subdirectory.

For example, to route messages and trace information to the default file ctg.trc in the CICS TG bin subdirectory, specify the following on the command line:

```
cd /opt/ctg/bin
ctgstart -trace -tfile
```

Gateway daemon trace can also be configured via the CTG.INI file. The Configuration Tool can be used to do this using **Tools -> Trace Settings**. The trace settings window will open as shown in Figure 9-7. To enable tracing, select the **Gateway** checkbox, in the Gateway trace file section.

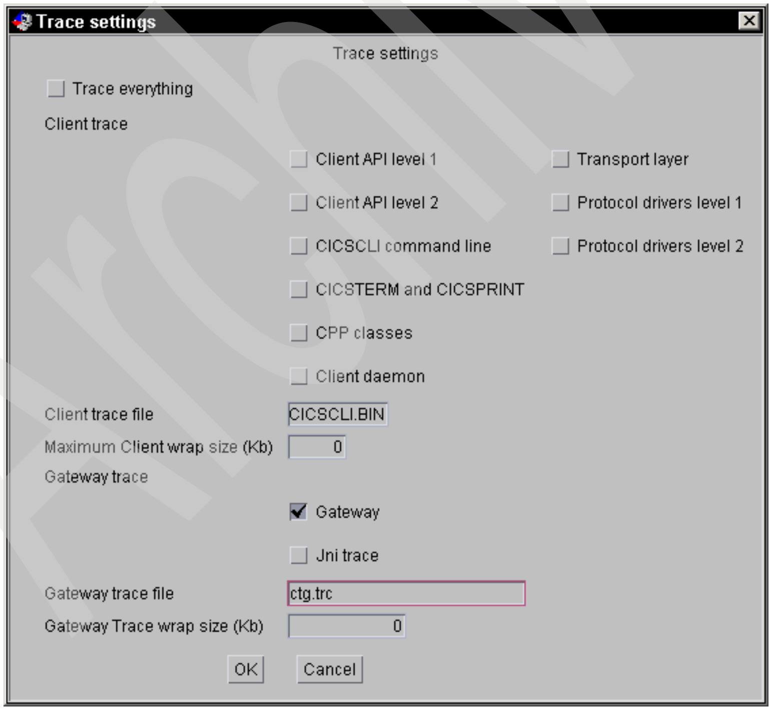


Figure 9-7 Configuration Tool trace settings

Dynamic trace

Gateway daemon trace can also be controlled dynamically during the Gateway daemon operation using the TCPAdmin protocol. This is a new function provided in the CICS TG V5.0. To enable the TCPAdmin protocol handler, select the **Enable protocol handler** check box for the TCPAdmin handler, as shown in Figure 9-8.

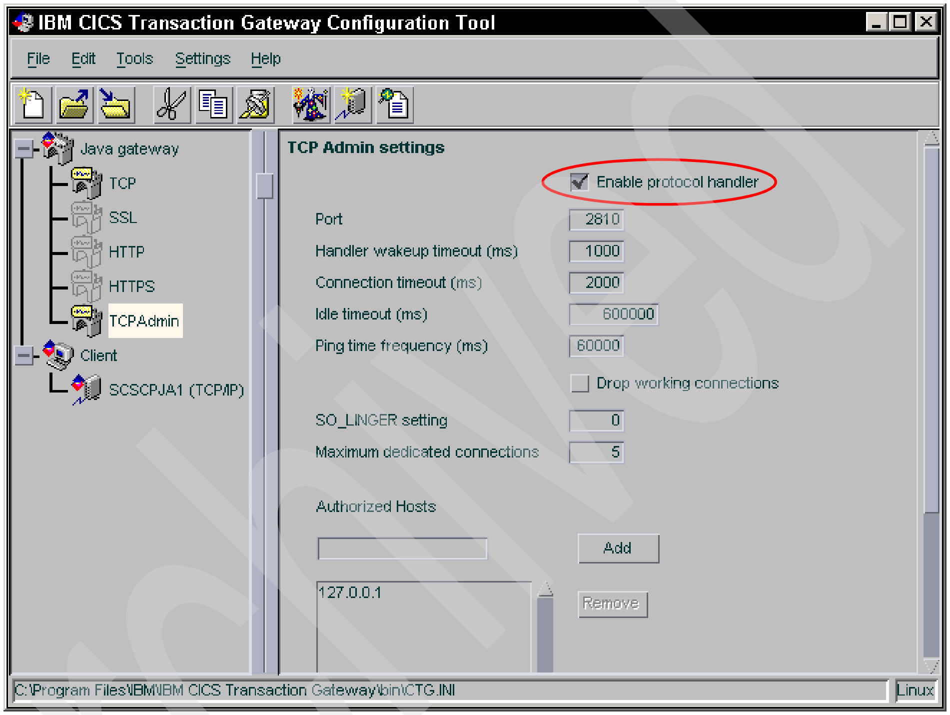


Figure 9-8 Configuring the TCPAdmin protocol handler

You can also set the authorized hosts that are allowed to control dynamic tracing by entering them in the Authorized Hosts field and clicking the **Add** button.

To activate the TCPAdmin protocol handler, save your changes and restart the Gateway daemon. Then you can use the tcpadmin.jar file from any authorized host by simply changing directory to the classes subdirectory, and calling the ctgadmin.jar as shown:

```
cd /opt/ctg/classes
java -jar ctgadmin.jar -ctg=tcp:\\wtsc66oe.itso.ibm.com:2810 -a=qtrace
```

For further details on how we used dynamic tracing see “Dynamic trace (gateway)” on page 174 in Chapter 7.

Gateway daemon JNI trace

Gateway daemon JNI trace is controlled at startup of the Gateway with the Java system property `gateway.T.setJINTFile=`<*pathname*>. This writes trace messages to the file specified in <*pathname*>. Java system properties can be passed to the Gateway daemon using the `-j` switch on the **`ctgstart`** command.

For example, to route JNI trace information to the ctgjni.trc file in the CICS TG bin subdirectory, specify the following on the command line:

```
cd /opt/ctg/bin
ctgstart -j-Dgateway.T.setJNITFile=ctgjni.trc
```

> **Tip:** You can also control JNI trace interactively using the TCPAdmin protocol handler, just as you can for Gateway daemon tracing.

Java client tracing

Java client tracing can be set from the command line when the application is invoked using the following Java directive:

```
java -Dgateway.T=on
```

or programmatically by adding the following statements to the Java application:

```
com.ibm.ctg.client.T.setDebugOn(true);
com.ibm.ctg.client.T.setTimingOn(true);
```

Both of these traces go to the stderr output stream, so they should be redirected to a file in order to save the trace output.

For applications using the local: CICS TG protocol, JNI tracing can be set from the command line when the application is invoked using the following Java directive:

```
java -Dgateway.T.setJNITFile=<pathname>
```

This writes trace messages to the file specified in <*pathname*>.

Client daemon

The CICS TG on Linux uses the services of the Client daemon to flow ECI and EPI requests to the connected CICS server. The Client daemon can be traced using the options specified in 5.4, “Problem determination” on page 90.

Linux error messages code file

Several messages in the Client daemon trace refer to return codes from functions in the C runtime library. We found the file /usr/include/asm/errno.h invaluable in determining what these codes mean. This file lists which message a return code corresponds to and a brief description of the cause for many system messages. For example, the entry in the Client daemon trace:

```
[5883] DRV:CCL5898 TCP62: recv failed, TCP/IP Rc= 107
```

shows that the socket function `recv` failed with return code 107. The definition for message 107 in errno.h is:

```
#define ENOTCONN        107     /* Transport endpoint is not connected */
```

This shows that a return code of 107 corresponds to the error ENOTCONN.

The manual page for the function `recv` gives further information on when this error might occur. This page can be accessed using the following command from a Linux command prompt:

```
man recv
```

This uses the **man** command to display the Linux programmer's manual pages for the item recv. For further uses of the **man** command, refer to the man manual pages, accessed using the command **man man**.

Part 4

WebSphere scenarios

In this section, we describe how we configured WebSphere Application Server V4 on Windows 2000 and z/OS to utilize the services of the CICS Transaction Gateway in order to communicate with a CICS Transaction Server region.

We describe both how to deploy a servlet-based Web application that uses the `ECIRequest` class provided in the CICS TG base classes, and how to deploy a J2EE based application that uses an integrated servlet/EJB design in conjunction with the CICS TG ECI resource adapter.

10

CICS TG and WebSphere Application Server for z/OS

In this chapter, we describe how we configured WebSphere Application Server V4.0.1 for z/OS and OS/390 to work together with the CICS Transaction Gateway (CICS TG) V5 for z/OS.

For details on how we installed the CICS TG for z/OS, refer to Chapter 7, "TCP connections to the Gateway daemon on z/OS" on page 133. For details on how we configured the required EXCI connection to our CICS region from the CICS TG, refer to 4.2, "EXCI connections" on page 67.

10.1 Preparation

We needed to install WebSphere Application Server V4, CICS TG V5, and CICS TS V2.2 on our z/OS system. In Figure 10-1, we show the components needed to support this configuration.

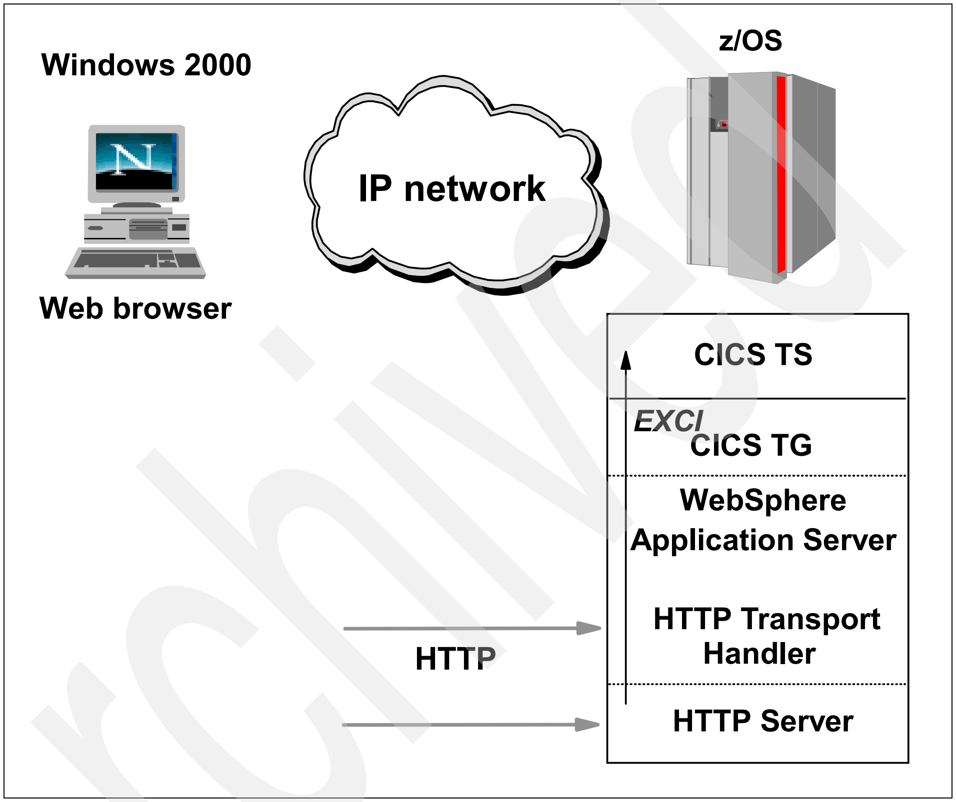


Figure 10-1 Software components: CICS TG and WebSphere for z/OS scenario

10.1.1 Software checklist

For this configuration we used the levels of software shown in Table 10-1.

Table 10-1 Software checklist: WebSphere for z/OS

Windows 2000 client	z/OS
Internet Explorer V6.0.2600.0000	z/OS V1.2
Windows 2000 Service Pack 2	WebSphere Application Server V4.0.1 at PTF level W401057 PTF, including including J2EE Connector function supplied by fix for APAR PQ55873
IBM WebSphere Application Server for z/OS and OS/390 Administration and Operations V4.01.014	IBM HTTP Server for z/OS V3.5
Application Assembly Tool (AAT) for z/OS V4.00.029	CICS Transaction Gateway for z/OS V5.00
	IBM Java Development Kit V1.3.1

10.1.2 Definitions checklist

The definitions we used to configure the scenario are listed in Table 10-2.

Table 10-2 Definitions checklist: WebSphere for z/OS

WebSphere z/OS	CICS TS	Example
hostname		wtsc66oe.itso.ibm.com
port		80 or 99
	APPLID	SCSCPJA1 or SCSCPJA4
WebSphere J2EE Server		C10ASR2
DFHJVPIPE variable	NETNAME defined in CICS CONNECTION definition	WASCTG

In addition, we also deployed our CICS COBOL application ECIPROG2 by compiling the source and placing the load module in a library in our CICS region's DFHRPL concatenation. We also enabled ASCII to EBCDIC data conversion by deploying a DFHCNV data conversion template in CICS for use by this program. Refer to Appendix A, "DFHCNV and CICS data conversion" on page 329 for more details. However, since our test applications (CTGTesterECI and CTGTesterCCI) can also convert the input and output data to and from EBCDIC, defining a data conversion template is optional in this case.

WebSphere configuration

We configured our WebSphere environment to use two different HTTP protocol catchers, the *HTTP Transport Handler* function of the WebSphere J2EE Server and the IBM *HTTP Server* (see Figure 10-2 on page 241).This means we had two different processes listening on two different ports, but both of them could forward HTTP requests into the Web container in the J2EE Server to run our applications. You will probably need to only use one of the protocol catchers but we had both configured for test purposes. The primary differences between the two protocol catchers are as follows:

- HTTP Transport Handler
  - Streamlined HTTP listener implemented as a function of the J2EE Server.
  - Does not currently support authentication either via HTTPS or via HTTP basic authentication.
- HTTP Server
  - Uses the HTTP Server address space in conjunction with the WebSphere plugin (as previously used in WebSphere Application Server V3.5).
  - Supports HTTP basic authentication, and HTTPS including client authentication.

Note: Since HTTP basic authentication was not currently supported with the HTTP Transport Handler at the time of writing this redbook, we used the HTTP Server environment to demonstrate how to flow the security context from the client through to our CICS region (see 10.3.2, "HTTP Server and basic authentication testing" on page 274).

However, subsequent to writing this book, basic authentication and HTTPS support (not including client authentication) was implemented by the J2EE Server in APAR PQ59911, and we advise you to use this function if you require Web client authentication.

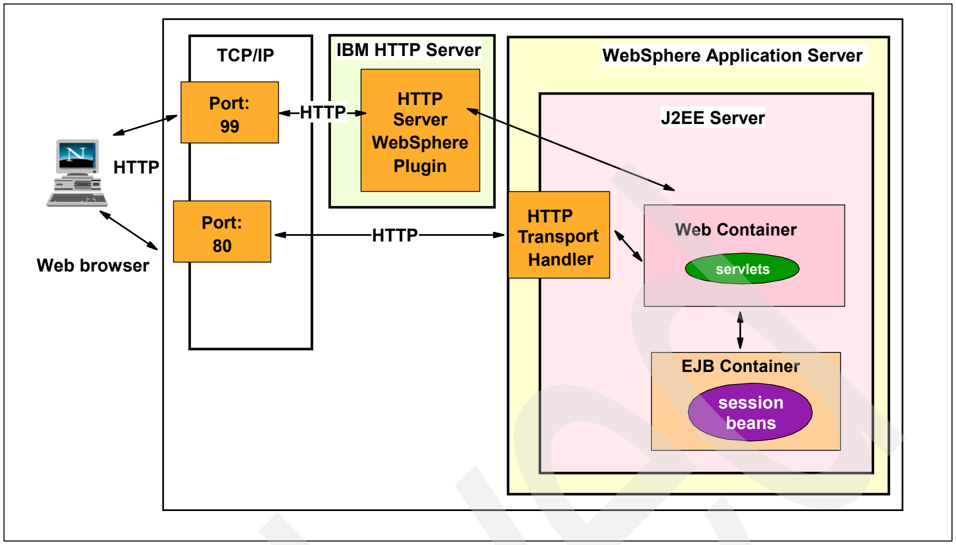


Figure 10-2 WebSphere configuration in the z/OS environment

In Figure 10-2, you can see in our configuration how HTTP requests to invoke Web applications can either be routed via port 99 and the HTTP Server or via Port 80 and the HTTP Transport Handler. However, in both cases the runtime environment is still the same.

For further details on the different configurations possible, we recommend the document *WebSphere Application Server V4.0 and V4.0.1 for z/OS and OS/390, Configuring Web Applications*, available from:

`http://www.ibm.com/support/techdocs`

Application structure

We used the following two Web applications, which we developed using WebSphere Studio Application Developer Integration Edition:

- A traditional servlet based application (*CTGTesterECI*), which uses only the CICS TG `ECIRequest` class to call a program in a connected CICS region. This runs in the Web container of WebSphere Application Server.
- A more modern J2EE application (*CTGTesterCCI*), which uses a servlet/session bean design to call a program in a connected CICS region. This application can either run in a managed or unmanaged environment. If it runs in a managed environment, the ECI resource adapter (as supplied by the CICS TG) must be installed into the application server.

For further details on how we developed these applications, refer to Appendix B, “Sample applications” on page 337.

CTGTesterECI

The CTGTesterECI enterprise application is a servlet-based Web application that calls the specified COMMAREA-based CICS program and displays the result of the returned COMMAREA (see Figure 10-3).

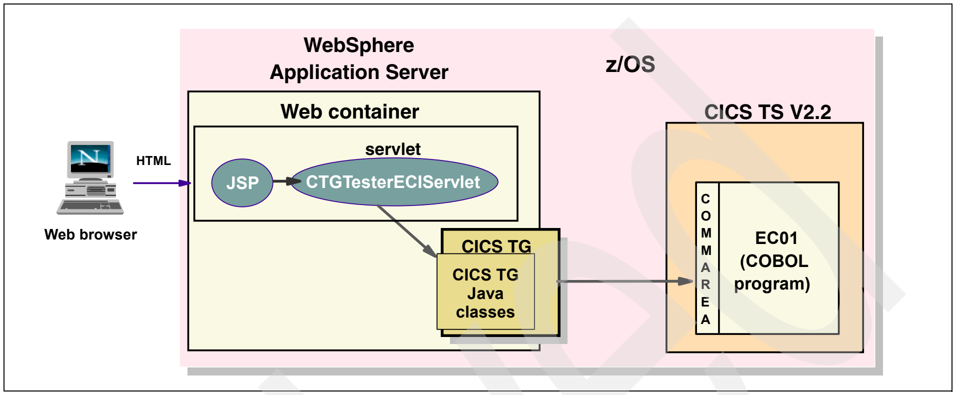


Figure 10-3 Application architecture of CTGTesterECI, local mode

The flow of control through the enterprise application is as follows:

- The JavaServer Page (JSP) index.jsp contains a form that starts the servlet CTGTesterECIServlet. Several parameters are sent to the servlet: the name of the CICS program to call, the encoding to use, and details of the Gateway daemon, among others.
- The servlet makes the call to CICS using the `ECIRequest` class from the CICS TG base classes. Once the call has completed, the servlet forwards the results to one of three JSPs, depending on whether the request was successful, an error occurred, or an exception occurred.
- The JSP formats and displays the results of the request.

This sample enterprise application is supplied in the file CTGTesterECI.ear. To obtain this file, see Appendix C, “Additional material” on page 389.

CTGTesterCCI

The CTGTesterCCI enterprise application calls a CICS program using the CICS ECI resource adapter and displays the result. Figure 10-4 illustrates the complete architecture of the enterprise application.

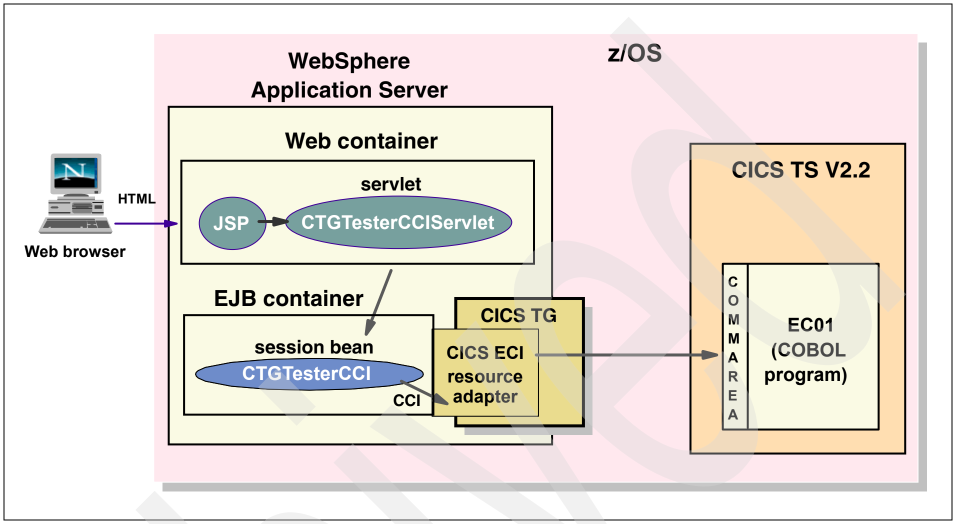


Figure 10-4 Application architecture of CTGTesterCCI, local mode

The flow of control through the enterprise application is as follows:

- The JSP page index.jsp contains an HTML form that starts the servlet CTGTesterCCIServlet. Several parameters are sent to the servlet, including the name of the CICS program to call, the COMMAREA input and length, and the encoding to use.
- The servlet passes these parameters on to the session bean, CTGTesterCCI.
- The session bean makes the ECI call to CICS using the CICS ECI resource adapter, and returns the response back to the servlet.
- The servlet forwards the results to one of two JSPs, depending on whether the request was successful or not.
- The JSP formats and displays the results of the request.

The servlet also allows use of an unmanaged connection, in which case several additional parameters, such as CICS server and user details, may be specified, since these are normally defined in the managed connection factory.

This sample enterprise application is provided in the file CTGTesterCCI.ear. To obtain this file, see Appendix C, “Additional material” on page 389.

10.2 WebSphere configuration

This redbook assumes you have a working WebSphere Application Server runtime environment. For more details on setting up such an environment, refer to the manual *WebSphere Application Server V4.0.1 for z/OS and S/390, Installation and Customization*, GA22-7834.

An essential part of using WebSphere for z/OS is the System Management End-User Interface, which consists of the *Operations* and *Administration* applications. Figure 10-5 shows our Operations application. For further details on configuring and using these tools, refer to “Installing the System Management End-User Interface” on page 248.

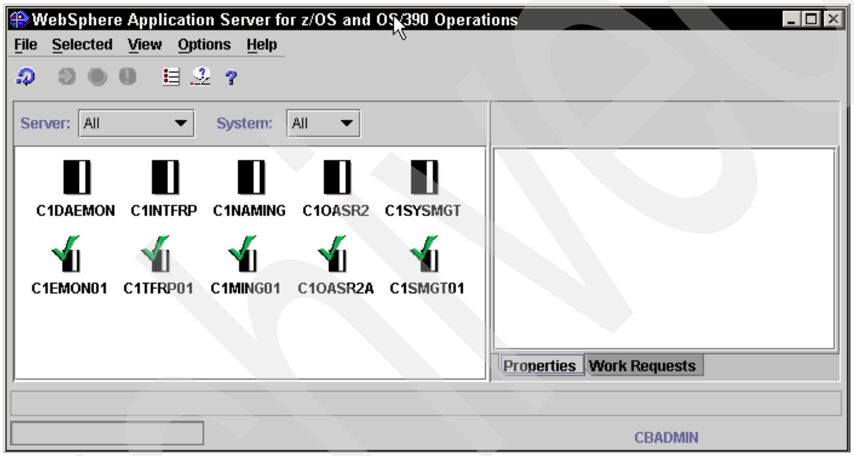


Figure 10-5 Operations application

The top row lists all the configured servers; the bottom row with the status icon ( ) lists the server instances. A server is a logical grouping of server instances, all server instances within a server having an identical structure.

In our configuration (Figure 10-5) we had the following server instances configured and active:

C1EMON01	Daemon server instance
C1TFRP01	Interface Repository server instance
C1MING01	Naming server instance
C1OASR2A	J2EE Server instance
C1SMGT01	System Management server instance

In addition to these server we also configured an LDAP server (C1OLDAP) and an HTTP Server (SCSWEBC6).

10.2.1 J2EE Server configuration files

A J2EE Server instance utilizes several different configuration files. The key files are as follows:

current.env	Current environment variables
jvm.properties	JVM properties settings
webcontainer.conf	Web container configuration

For our J2EE Server instance, these files were located in the directory:

/WebSphereC1/CB390/controlinfo/envfile/WTSCPLX1/C1OASR2A/

In Example 10-1 we show the content of our jvm.properties file for our J2EE Server instance. These settings were created from the settings input using the Administration application. After changes are *activated* using the Administration application, the settings are written to the jvm.properties file.

Example 10-1 jvm.properties for J2EE Server instance

```
com.ibm.ws390.wc.config.filename=/WebSphereC1/CB390/controlinfo/envfile/
WTSCPLX1/C1OASR2A/webcontainer.conf
com.ibm.ws390.trace.settings=/WebSphereC1/CB390/controlinfo/envfile/WTSCPLX1/
C1OASR2A/trace.dat
com.ibm.ws390.server.classloadermode=1
```

In Example 10-2, we show the content of our current.env file for our J2EE Server instance C1OASR2A. These settings were created from the settings input using the Administration application. After changes are *activated* using the Administration application, the settings are written to the current.env file.

Example 10-2 current.env for J2EE Server instance

```
# ENVIRONMENT FILE FROM CONVERSATION 20.6. 10:24
#----------------------------------------------
CLASSPATH=/usr/lpp/WebSphere/lib/ws390srt.jar:/usr/lpp/WebSphere/lib/xerces.jar
:/usr/lpp/WebSphere/lib/waswebc.jar:/usr/lpp/WebSphe re/lib/bboaxrt.jar:
/usr/lpp/db2/db2710/classes/db2j2classes.zip:/usr/lpp/WebSphere/lib/
waswebccp.jar:/usr/lpp/ctg500/ctg/deployable/cicseciRRS.jar:/usr/lpp/ctg500/ctg
/deployable/cicsframe.jar:/usr/lpp/ctg500/ctg/deployable/ctgserver.jar:/usr/lpp
/ctg500/ctg/deployable/ctgclient.jar:
JVM_LOGFILE=/tmp/C1OASR2.jvm.log
BBOC_HTTP_PORT=80
BBOC_HTTP_LISTEN_IP_ADDRESS=9.12.6.29
LIBPATH=/usr/lpp/db2/db2710/lib:/usr/lpp/java/IBM/J1.3/bin:/usr/lpp/java/IBM/
J1.3/bin/classic:/usr/lpp/WebSphere/lib:/usr/lpp/ctg500/ctg/deployable
MAX_SRS=5
DFHJVPIPE=WASCTG
TRACEBUFFLOC=SYSPRINT
```

```
BBOLANG=ENUS
CBCONFIG=/WebSphereC1/CB390
DAEMON_PORT=5555
DAEMON_SSL_PORT=5556
DATASHARING=1
DM_GENERIC_SERVER_NAME=C1DAEMON
DM_SPECIFIC_SERVER_NAME=C1EMON01
ICU_DATA=/usr/lpp/WebSphere/bin
IR_GENERIC_SERVER_NAME=C1INTFRP
IR_SPECIFIC_SERVER_NAME=C1TFRP01
IRPROC=C1OIR
IVB_DRIVER_PATH=/usr/lpp/WebSphere
LDAPCONF=/WebSphereC1/CB390/WTSCPLX1/etc/ldap/SC66.bboslapd.conf
LDAPIRCONF=/WebSphereC1/CB390/WTSCPLX1/etc/ldap/SC66.bboslapd.conf
LDAPIRROOT=o=BOSS,c=US
LDAPROOT=o=BOSS,c=US
LOGSTREAMNAME=WAS.SC66.ERROR.LOG
NM_GENERIC_SERVER_NAME=C1NAMING
NM_SPECIFIC_SERVER_NAME=C1MING01
NMPROC=C1ONM
OTS_DEFAULT_TIMEOUT=300
OTS_MAXIMUM_TIMEOUT=300
RAS_MINORCODEDEFAULT=NODIAGNOSTICDATA
RESOLVE_IPNAME=wtsc66oe.itso.ibm.com
RESOLVE_PORT=900
SM_DEFAULT_ADMIN=CBADMIN
SM_GENERIC_SERVER_NAME=C1SYSMGT
SM_SPECIFIC_SERVER_NAME=C1SMGT01
SMPROC=C1OSMS
SOMOOSQL=1
SRVIPADDR=9.12.6.29
SYS_DB2_SUB_SYSTEM_NAME=DB7E
TRACEALL=1
TRACEPARM=00
DB2SQLJPROPERTIES=/WebSphereC1/CB390/db2sqljjdbc.properties
DAEMON_IPNAME=wtsc66oe.itso.ibm.com
CONFIGURED_SYSTEM=SC66
```

Tip: Environment variables can be set at different levels with WebSphere Application Server. Since we had only one J2EE Server and one J2EE Server instance, this was of little concern in our configuration. However, in a more complex environment you should consider setting common environment variables at the highest level.

In the case of the CICS TG, we suggest you set the CLASSPATH and LIBPATH at the J2EE Server level, and DFHJVPIPE and at the J2EE Server instance level. You should also note that there is no need to set CTG_RRMNAME when using the CICS TG with WebSphere for z/OS, since WebSphere performs the registration with MVS Resource Recovery Services (RRS).

10.2.2 HTTP Server definitions

We added the following statement to our HTTP Server configuration file /web/cics6/httpd.conf:

```
Service      /CTGTester*/*
/usr/lpp/WebSphere/WebServerPlugIn/bin/was400plugin.so:service_exit
```

This statement causes inbound HTTP requests starting with the string `/CTGTesterECIWeb/*` to be routed to the WebSphere plugin.

10.2.3 System Management End-User Interface

This section describes the steps you have to take to install and run the System Management End-User Interface on your workstation.

Establish communication to the host

The Administration and Operations applications both use TCP/IP to communicate between your workstation and the host. All IP names used to communicate with the host must be defined either to your DNS server or in your workstation hosts file. The host names that are required are as follows:

- The bootstrap server IP name

  The name associated with the initial connection to the host. It is defined by the RESOLVE_IPNAME parameter of the z/OS sysplex environment file (as shown in Example 10-2 on page 245).

- The naming server IP name

  A generic name associated with your naming server and defined by the DAEMON_IPNAME parameter of the z/OS sysplex environment file. If you have more than one name server (that is, a federated name space) you must ensure that all the name servers' host names needed by the workstation can be resolved.

- The host name of each system in the sysplex in which WebSphere for z/OS runs.

Installing the System Management End-User Interface

After the Application Server is installed on z/OS, you can download the System Management End-User Interface via FTP as a binary file. The file bboninst.exe is supplied on the host in the path /usr/lpp/WebSphere/bin, which you should download to your workstation and use to install the tool.

Defining workstation environment variables for login options

The BBONPARM workstation environment variable is used to pre-assign various login options for the Administration and Operations applications. These values may be overwritten when you start the Administration or Operations application. The login options are:

bootstrapserver <nameserver>	Names the default IP name for the naming server
bootstrapport <port>	Names the port used to connect to the naming server
loginuser <user ID>	Names the default user ID for the login
loginpassword <password>	Names the default password for the login

On our workstation, we did as follows to set the bootstrap server and user ID:

1. Open the Windows control panel by clicking **Start -> Settings -> Control Panel**. Then select **System -> Advanced -> Environment Variables -> User variables for cicsrs1**, where cicsrs1 is your Windows user ID.
2. Define a new variable BBONPARM
3. Enter the value `-bootstrapserver wtsc66oe.itso.ibm.com -loginuser CBADMIN -loginpassword XXXXXXX`
4. Click **OK** and **OK** to set the new variable.

These default values will now be used as the default in the login panel the next time the application is started. See Figure 10-6 on page 249.

Starting the Administration or Operations application

To start the Administration or Operations application from your workstation, do as follows:

1. Select **Start -> Programs -> IBM WebSphere for z/OS -> Administration** or **Operations** as required.

2. This will display the Login window as shown in Figure 10-6.
3. Modify the values as appropriate and then click **OK** to log in.

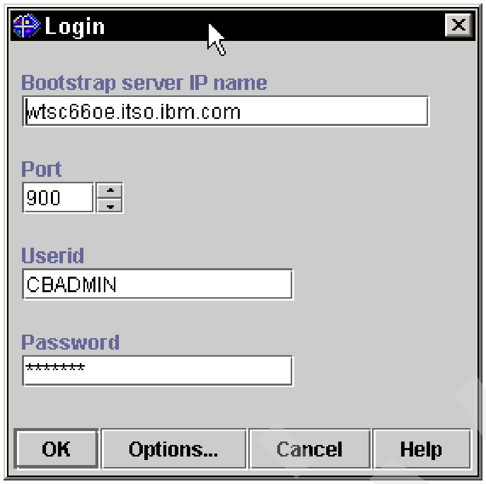


Figure 10-6 Login window

This will now start either the Administration application (as shown in Figure 10-7) or Operations application (as shown in Figure 10-5 on page 244).

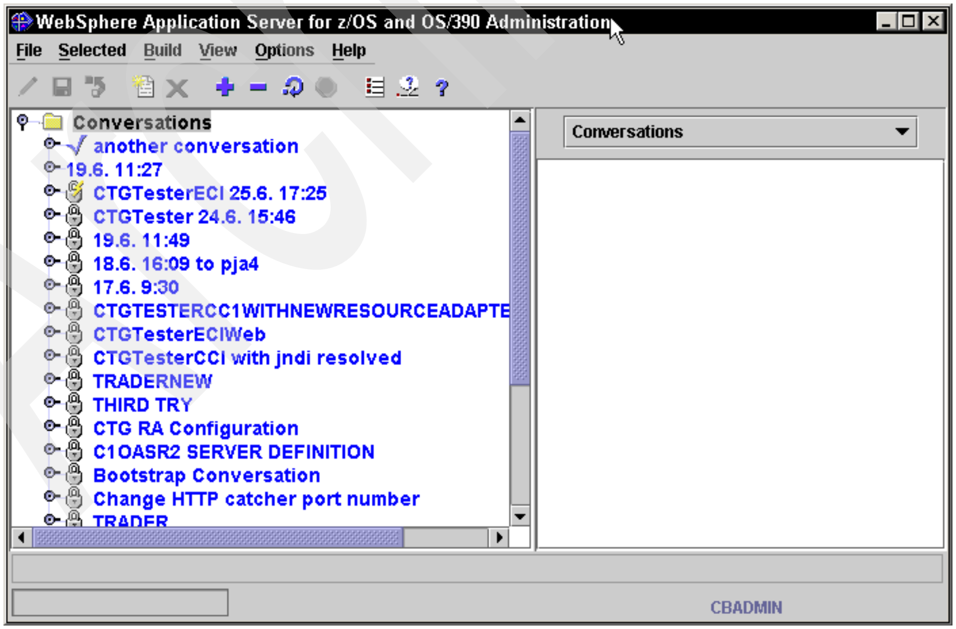


Figure 10-7 Administration application

10.2.4 Installing the CICS ECI resource adapter

This section describes the steps necessary to prepare for the use of the CICS ECI resource adapter with WebSphere Application Server. It consists of the following steps:

- CICS TG considerations
- CICS TS considerations
- Installation of the CICS ECI resource adapters

CICS TG considerations

Since the CICS ECI resource adapter uses the functions of the CICS TG to connect to CICS, you will need to ensure the CICS TG environment is fully working before it can be used by WebSphere for z/OS. However, you do not need to start the Gateway daemon, since WebSphere will use the CICS TG in local mode, whereby all the CICS TG Java classes are executed within the WebSphere address space. The key points to consider are as follows:

- Mark UNIX System Services programs as program controlled with the **`extattr +p`** command.
- Allow the CICS TG user ID read access to program controlled libraries.

Further details can be found in Chapter 7, "TCP connections to the Gateway daemon on z/OS" on page 133.

CICS TS considerations

You need to perform the following tasks:

- Configure an EXCI connection in CICS
- Set the RRMS, and IRCSTRT SIT parameters
- Define the CICS EXCI library to your J2EE Server
- Define RACF security profiles

SIT parameters

Since the CICS TG and WebSphere use EXCI for communications and MVS resource recovery services (RRS) for transactional support, you should enable support in CICS using the following SIT parameters:

- RRMS=YES
- IRCSTRT=YES
- ISC=YES

You will then need to restart your CICS region to put these parameters into effect.

EXCI connections

Because the CICS ECI resource adapter uses the function of the CICS TG to connect to CICS, you will need to configure an EXCI CONNECTION definition in all the CICS regions with which your J2EE Server whishes to communicate. Full details on how to configure EXCI connections are supplied in Chapter 4, “EXCI connections to CICS” on page 65.

The main choice you must make is to decide if you will use a *generic* or *specific* EXCI pipe to communicate with CICS. The default is to use a generic pipe (which is basically an unamed pipe). If you wish to use a specific pipe you must define the DFHJVPIPE environment variable in the J2EE Server instance that will be used for the CICS connector support. This value must match the NETNAME parameter of an installed EXCI CONNECTION definition in CICS.

We recommend using specific EXCI pipes, as this provides for better monitoring and accounting within CICS. The same specific pipe can still be used by multiple J2EE Server instances, to connect to the same CICS region so this does not limit your ability to work with multiple J2EE Servers in a workload management environment.

Defining the EXCI library to your J2EE Server

The J2EE Server that will use the CICS ECI resource adapter needs to access and load the CICS module DFHXCSTB from the SDFHEXCI library supplied with CICS. This library will typically be called CICSTS13.CICS.SDFHEXCI or CICSTS22.CICS.SDFHEXCI, depending on the CICS release and naming conventions you use.

To enable this you will need to complete one of the following steps:

- Add module DFHXCSTB to the LPA.
- Add the SDFHEXCI library to either the system linklist concatenation, or to the STEPLIB concatenation of the J2EE Server.

We added our SDFHEXCI data set CICSTS22.CICS.SDFHEXCI to the STEPLIB in the startup procedure of the J2EE Server instance.

Surrogate security profiles

As a user of the CICS TG and the EXCI, the J2EE Server will be subject to RACF surrogate security checking. If surrogate security checking is enabled in DFHXCOPT, the EXCI options table, the user ID of the EXCI job (in our case the user ID of the J2EE Server region) will require read access to the *USERID*.DFHEXCI SURROGAT class profile, where *USERID* is the user ID flowed in the EXCI request. This authorizes the J2EE Server user ID to act as a surrogate for the user ID specified by the clients that may invoke J2EE applications. We defined the following profile, as we wished to give a universal access of READ to all users (in effect disabling surrogate security) for testing purposes:

```
RDEFINE SURROGAT *.DFHEXCI UACC(READ)OWNER (CICSRS1)
```

```
SETROPTS RACLIST(SURROGAT) REFRESH
```

Installation of the CICS ECI resource adapter

Resource adapters are packaged in Resource Adapter Archive (RAR) files, which like EAR files are archives made up of JAR files and a deployment descriptor. The CICS ECI resource adapter for z/OS is provided in the file cicseciRRS.rar in the deployable directory of the CICS TG installation.

This cicseciRRS.rar resource adapter archive is designed exclusively for z/OS. It makes use of the two-phase commit capability of RRS to provide support for global transactions. All non-z/OS platforms use the cicseci.rar file instead.

Unlike WebSphere Application Server Advanced Edition, WebSphere Application Server for z/OS does not work with RAR files directly, so you must extract the contents of the RAR file into a directory, and point the WebSphere classpath to each JAR file in this directory.

To install the CICS ECI resource adapter, we entered the following commands under OMVS:

- `export PATH=/usr/lpp/java/IBM/J1.3/bin:$PATH`

  This adds the bin directory of our Java1.3 installation to the PATH environment variable.
- `cd /usr/lpp/ctg500/ctg/deployable`

  This changes directory to the location where the CICS TG supplies the cicseciRRS.rar file.
- `jar -xvf cicseciRRS.rar`

  This extracts the following files, as shown in Example 10-3.

Example 10-3 Extract of cicseciRRS.rar

```
$ SC66¨ /usr/lpp/ctg500/ctg/deployable: jar -xvf cicseciRRS.rar
  created: META-INF/
extracted: META-INF/MANIFEST.MF
extracted: META-INF/ra.xml
extracted: ccf2.jar
extracted: ctgclient.jar
extracted: ctgserver.jar
extracted: cicseciRRS.jar
extracted: cicsframe.jar
extracted: libCTGJNI.so
extracted: libCTGJNI_g.so
```

We then used the following commands to add execute permissions for all users to the CICS TG JNI shared library (libCTGJNI.so):

```
chmod ugo+x libCTGJNI.so
```

Defining connection information

You are now ready to define connection information to the J2EE Server. We performed the following steps:

1. Start the WebSphere Application Server for z/OS and OS/390 Administration application using the menu option: **Start -> Programs -> IBM WebSphere for z/OS -> Administration**.
2. Log on to your server using its IP address, port, your user ID, and password. We used `wtsc66oe.itso.ibm.com, port 900,` and `CBADMIN` as our user ID, as shown in Figure 10-6 on page 249.
3. Create a new conversation to add the CICS connector support. To do this, right-click **Conversations** and select **Add**.

> **Tip:** We suggest you use a date/time based convention for your conversation names, since we found you will need to define a new conversation every time you make a change to WebSphere. We used the format <application> <day>.<month> <time>, for example *CTGTester 24.5 15:45*. You will also see this same conversation name at the top of your current.env file (which is the file WebSphere uses to store your current definitions) after this conversation is activated. With this method, you will then know when you made the changes for the active WebSphere configuration.

4. Enter the conversation name and then click **Selected -> Save.** Our new conversation *24.5 15:45* is shown in Figure 10-8.

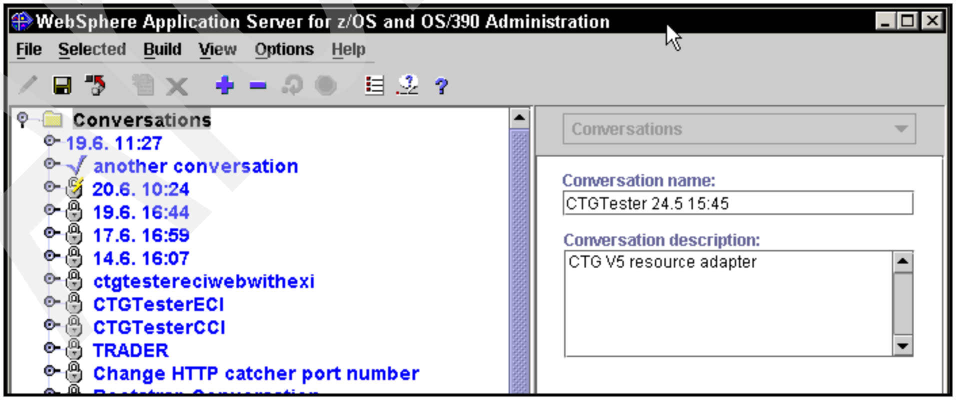


Figure 10-8 Entering the conversation name

Important: You might also be tempted to modify the current.env file instead of using the Administration application for updating J2EE Server definitions, since this is a lot quicker than using the Administration application. If you do update current.env, the update will last only until the next conversation is activated and your change will not be input to the next conversation. However, this is still a very useful method for quickly making minor or temporary changes, such as settings trace parameters or modifying the CLASSPATH.

5. Now turn on connection management in the sysplex. To do this, expand the tree down to the sysplex level (in our case, click **24.5 15:45 -> Sysplexes -> WTSCPLX1**). Then right-click the sysplex name and select **Modify**. In the Configuration Extensions section, check the box labeled **Connection Management**. Click **Selected -> Save** to save your changes.
6. Next, expand the list of J2EE Servers and locate the J2EE Server that you wish to use with the CICS ECI resource adapter. Right-click the J2EE Server and select **Modify**. See Figure 10-9.

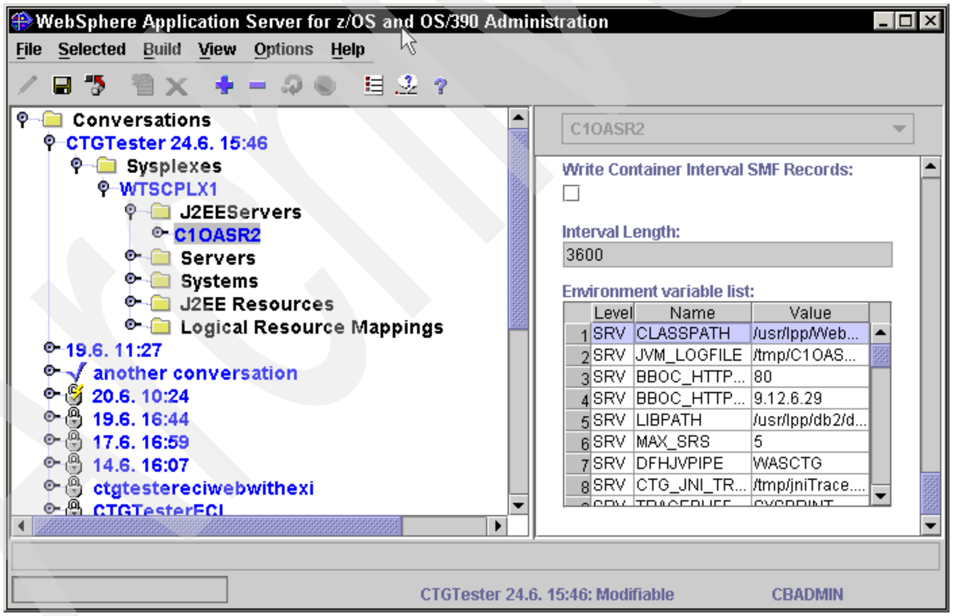


Figure 10-9 Updating J2EE Server definitions

7. Make the following changes in the environment variable list:

a. Add the following JAR files in the /usr/lpp/ctg500/ctg/deployable directory to the CLASSPATH. These are the JAR files that you created earlier from the cicseciRRS.rar file:

- cicseciRRS.jar
- cicsframe.jar
- ctgserver.jar
- ctgclient.jar

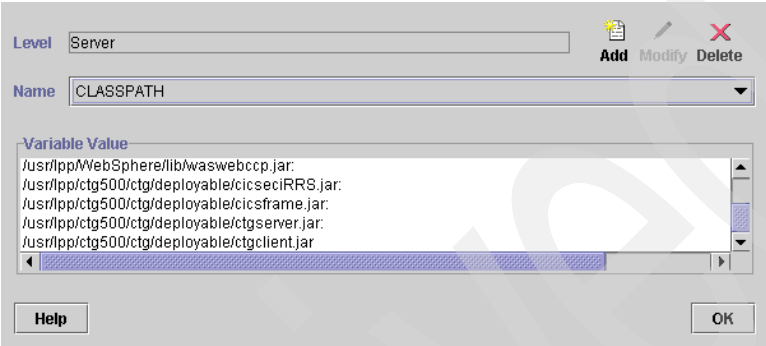


Figure 10-10 Updating CLASSPATH

b. Add the connectors directory to the LIBPATH. Our directory was /usr/lpp/ctg500/ctg/deployable.

8. Click **Selected -> Save** to save your changes.

9. To define a CICS ECIConnectionFactory as a new J2EE resource perform the following:

– Locate the J2EE Resources folder under your sysplex, right-click it and select **Add**.

– In the properties window, enter the following:

- Set the J2EE resource name. We used `SCSCPJA1`.
- Set the J2EE resource type to `CICS_ECIConnectionFactory`.

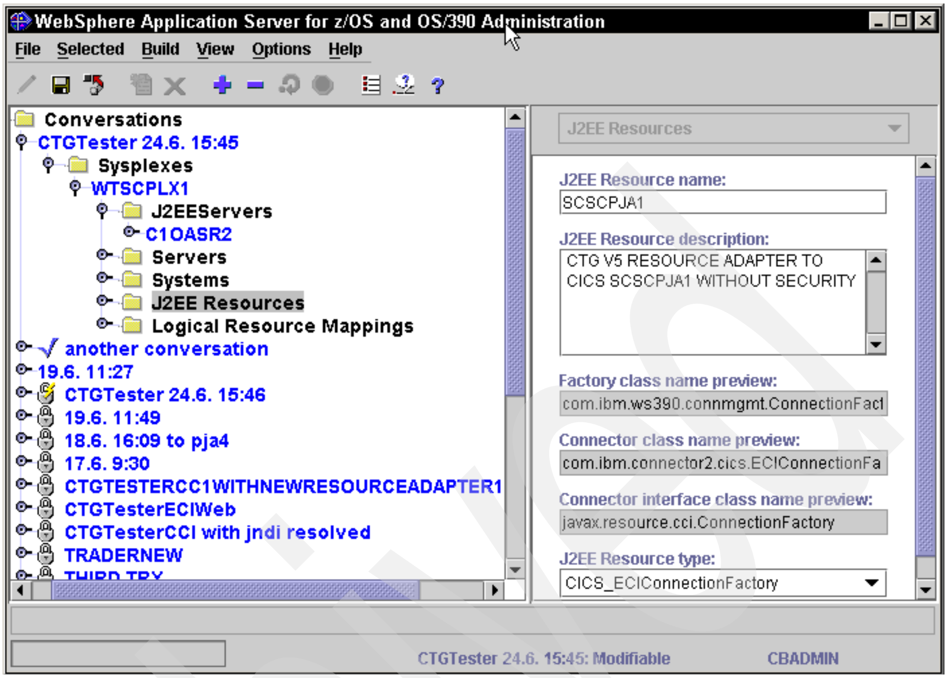


Figure 10-11 J2EE resource SCSCPJA1

10. You can now create a J2EE resource instance that defines the connection information. To do this, perform the following:

   – Expand the J2EE resource you have just created. This displays a J2EE resource instances folder. Right-click it and select **Add**.

   – In the properties window, enter the following:

      • Set the CICS_ECIConnectionFactory instance name. We used `SCSCPJA1`.

      • Select the System name with which this J2EE resource instance is to be associated. We used `SC66`.

      • Set the Server name to the APPLID of the CICS server you wish to call. We used `SCSCPJA1`.

   – Click **Selected -> Save** to save your changes. Look for the following message in the status bar to confirm the operation completed successfully:

```
BBON0515I J2EE resource Instance [name ] was added..
```

Tip: The Administration program also has a message log where you can check these messages. Click **File -> Message log**.

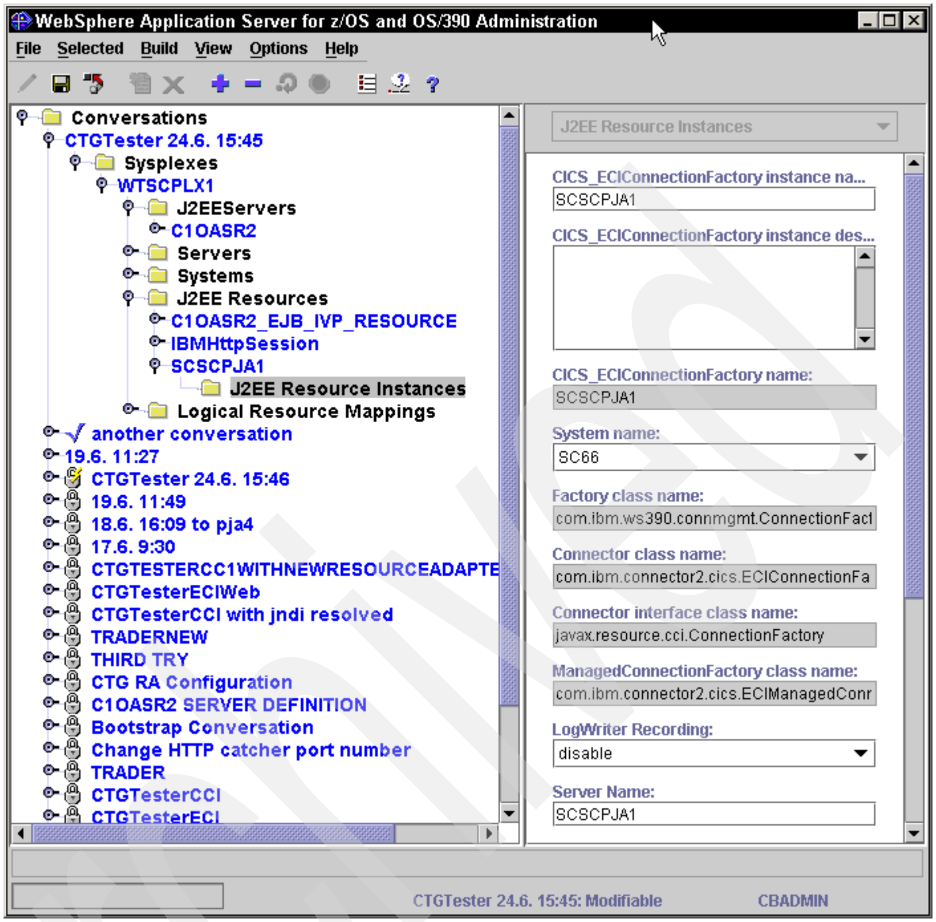


Figure 10-12 J2EE resource instance SCSCPJA1

> **Note:** We let the other fields in the CICS ECI connection factory default. However, you can set the user ID and password in the connection factory, but they will be used only if they are *not* provided by either the application or the container. You can also specify the TranName or TPNName, which can be used to modify the name of the CICS mirror transaction from the default, which will be CSMI.

The J2EE Server region is now configured to use the CICS ECI resource adapter. We defined a resource adapter for both of our CICS regions (SCSCPJA1 and SCSCPJA4). To do this, you need to repeat steps 9 and 10 to build multiple resource adapters. Do not commit the conversation yet because you still need to deploy an application to make use of this support. This is described in the next section.

10.2.5 Deploying our sample enterprise applications

EAR files generated in WebSphere Studio cannot be directly deployed to WebSphere Application Server for z/OS. You must use the Application Assembly Tool (AAT) for z/OS to generate an EAR file that WebSphere can use. Using the AAT for z/OS also gives you the opportunity to change some deployment information. We downloaded the AAT for z/OS from the following Web site:

`https://www6.software.ibm.com/dl/websphere20/zosos390-p`

In this section, we used our sample J2EE enterprise application CTGTesterCCI. The source code for this application is provided in the additional material for this book. See Appendix C, “Additional material” on page 389.

Deploying the enterprise application

1. Start the AAT for z/OS by clicking **Start -> Programs -> IBM WebSphere for z/OS -> Application Assembly**. In the left pane, right-click **Applications** and select **Import**. Specify the location of the CTGTesterCCI.ear, and click **OK**. The file should import successfully.

   Next, you will probably receive a few pop-up windows warning about missing classes (as shown in Figure 10-13). The first one of these was for `javax.resource.ResourceException`, to which we answered **Yes**. We were then required to select the JAR file containing this class (connector.jar) to be added to our EAR file. This is because our application was designed to throw a ResourceException and so this class is needed to generate the deployment code. To all the other questions we replied **No**, because we wanted to have the classes taken from the WebSphere runtime environment, as otherwise we found that we could have class loader problems.

> **Important:** It is possible to circumvent these errors by adding the JARs to the variable USER_AAT_CLASSPATH.

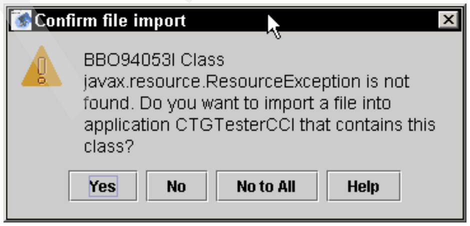


Figure 10-13 AAT file import

2. Under the applications folder, there should now be a CTGTesterCCI application. Fully expand it to show all the components contained within the application, as shown in Figure 10-14.

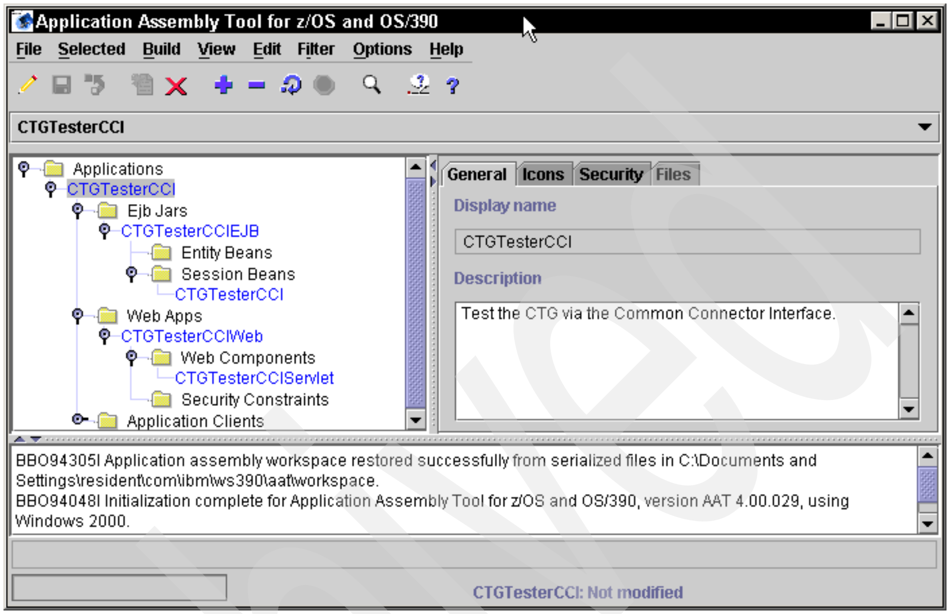


Figure 10-14 AAT showing CTGTesterCCI application

3. Click **CTGTesterCCI -> Ejb Jars -> CTGTesterCCIEJB -> Session Beans -> CTGTesterCCI**.

   From here you can set deployment descriptor information for the session bean. For the purposes of this test, we will let most things default or leave them to keep the same values as we set in WebSphere Studio. However, here you could change the attributes, if needed. We clicked **Transactions -> Transaction attribute -> Required** because we wanted to use the CICS TG two-phase commit support to join the CICS and WebSphere transactions. You can also modify the security parameters `RunAs` and `ThreadID`. We did not modify these parameters at this time, but for further details refer to "Security settings" on page 280.

4. We now need to connect our Web application CTGTesterCCIWeb to our session bean. Our servlet is coded so that it will do a JNDI lookup for the bean using the EJB reference, but we have to give it the following information:

   – First select the Web application by clicking **Applications -> CTGTesterCCI -> Web Apps -> CTGTesterCCIWeb**.

   – Click **Selected -> Modify** and then select the **EJBs** tab.

– Select the Reference Name **ejb/CTGTesterCCI** then click **Modify**.

– In the Link drop-down box, select **CTGTesterCCI** and then click **OK**. This screen is shown in Figure 10-15.

– Last, save these changes by clicking **Selected -> Save**.

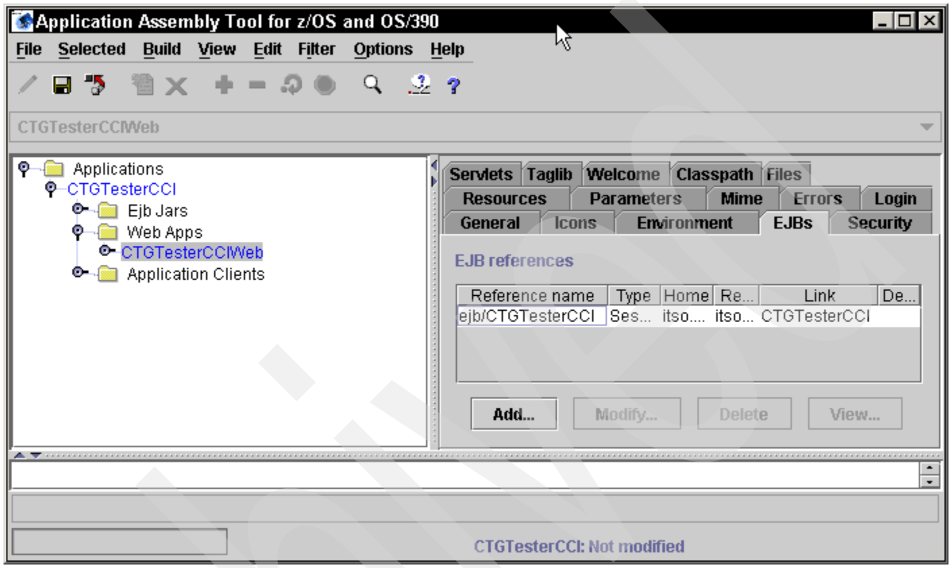


Figure 10-15 EJB reference

5. You are now ready to create the deployment code for CTGTesterCCI. Select the application **(CTGTesterCCI)** from the Applications node. Then from the menu bar select:

   a. **Build -> Validate**

   b. **Build -> Deploy**

   Click **No** to the prompts to add required classes. Look for the following message to confirm deployment has successfully completed:

   ```
   BBO94009I Application CTGTesterCCI was deployed.
   ```

6. Finally, export the modified CTGTesterCCI application from the AAT for z/OS to your local machine. To do this:

   – Click the **CTGTesterCCI** application and select **Export**.

   – Enter a path to store the EAR file, and be sure that the **WebSphere for z/OS Version 4.0 compatible** option is checked.

   – Click **OK**.

– Look for the following message to confirm export has successfully completed:

```
BBO94011I Application CTGTesterCCI was exported to EAR file
C:\itscctgv5\CTGTesterCCI_Deployed.ear.
```

Installing the J2EE application

The final task is to take the EAR file generated in the AAT for z/OS, and deploy it to a J2EE Server in WebSphere. Complete the following steps:

1. Return to the conversation you were previously editing in the System Management End-User Interface. Right-click the J2EE Server where you installed the CICS ECI resource adapter and select **Install J2EE Application**.
2. In the EAR Filename field, enter the name of the EAR file you exported from the AAT for z/OS, then click **OK**.
3. The Reference and Resource Resolution window will open, where you will need to set and resolve some references before the CTGTesterCCI enterprise application can be deployed.
   a. Expand the **CTGTesterCCI EJB** folder and highlight the **CTGTesterCCI** session bean. There are several properties we need to set in here.

      Select the **EJB** tab. From here you can specify the full JNDI name to give the CTGTesterCCI session bean. You can set this value to whatever you want, since our Web application does not hard code the JNDI name of the session bean. We clicked **Set Default JNDI Path & Name** to use the defaults (Figure 10-16 on page 262).

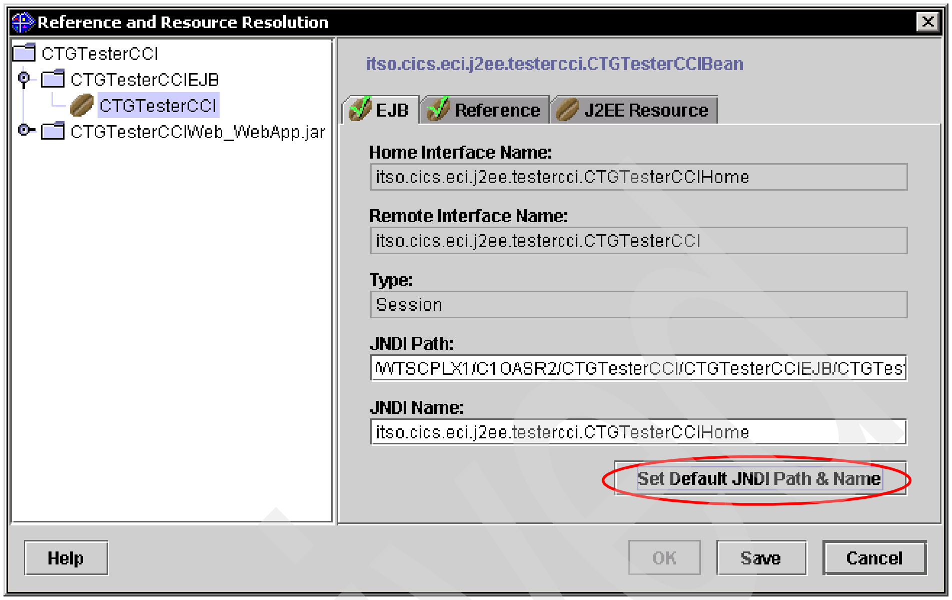


Figure 10-16 CTGTesterCCI bean, EJB tab

b. Next click the **J2EE Resource** tab. Here you associate the resource reference used by the session bean with an ECIConnectionFactory defined to the J2EE Server. Click in the blank J2EE resource field, and select the resource you created earlier. We selected **SCSCPJA1**. Your windows should now look as shown in Figure 10-17.

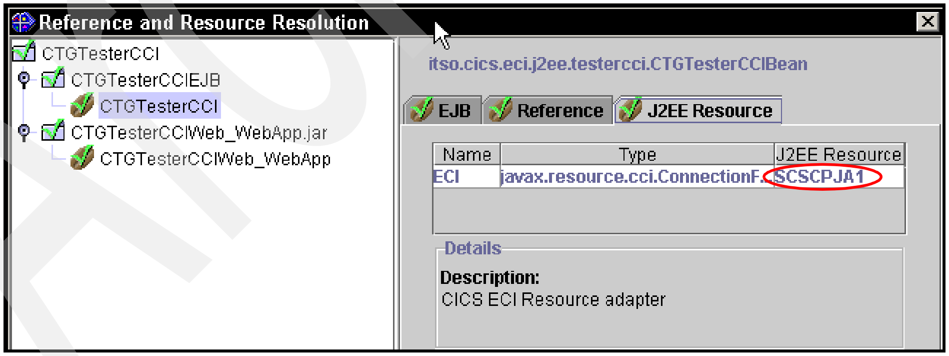


Figure 10-17 CTGTesterCCI bean, J2EE Resource tab

4. Now expand the **CTGTesterCCIWeb_WebApp.jar** folder and select the Web application **CTGTesterCCIWeb_WebApp**. Here you are required to provide a JNDI name for the Web application. This is used internally by WebSphere,

and does not affect our application, so we clicked **Set Default JNDI Path & Name**, which filled in the default values, shown in Figure 10-18.

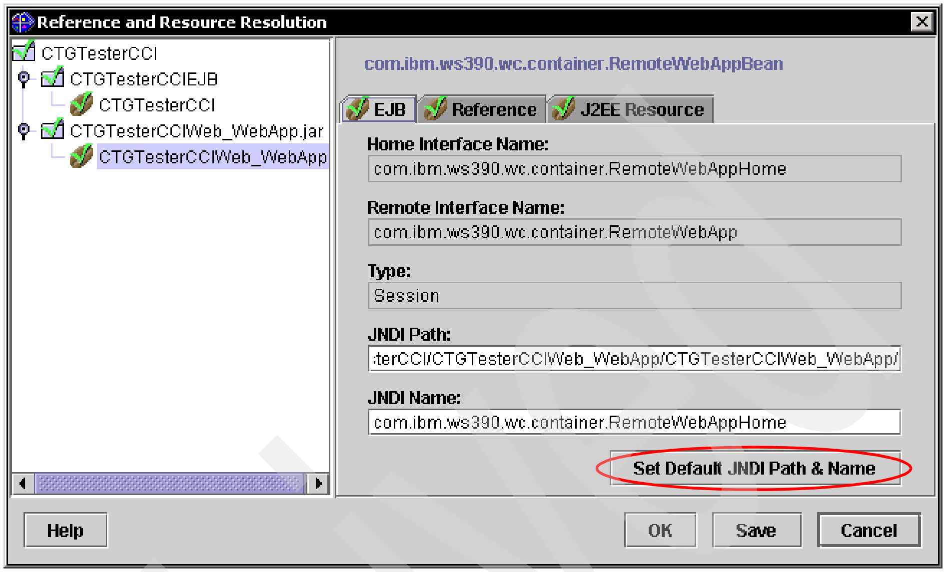


Figure 10-18 CTGTester Web application, EJB tab

5. If you click the **Reference** tab, you will also see the resolved link to the session bean (Figure 10-19 on page 264). This is the information we created earlier using the AAT for z/OS, as shown in Figure 10-15 on page 260.

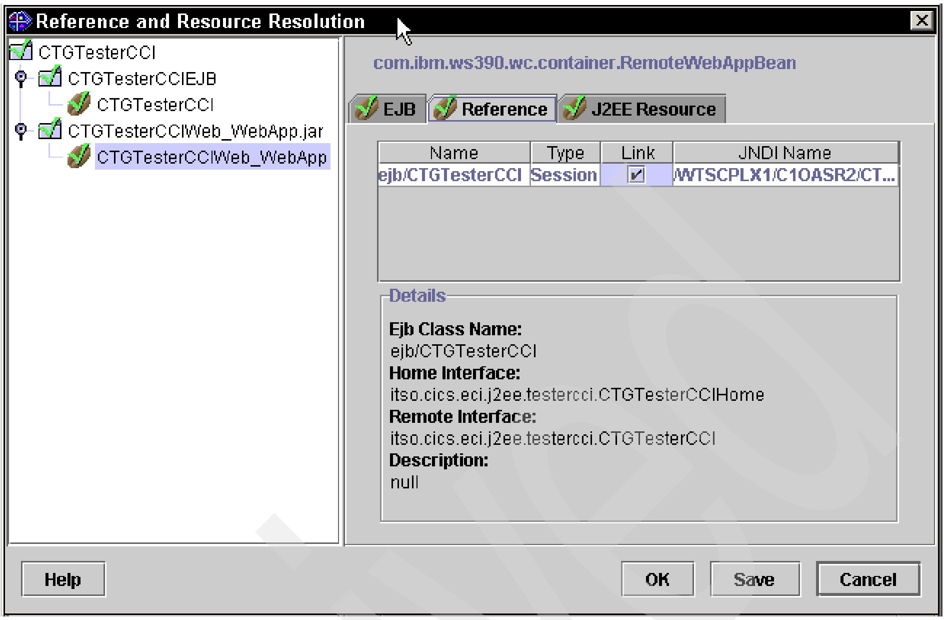


Figure 10-19 CTGTester Web application, Reference tab

6. All required references should have now been set, and the OK button will become active. Click **OK** to deploy the enterprise application. If the operation completes successfully, a message similar to the following will display:

```
BBON0470I EAR file CTGTesterCCI_deployed_resolved.ear has been
successfully installed on server C1OASR2.
```

7. You are now ready to activate the conversation. Right-click the conversation name (**CTGTester 24.6. 15:45**) and complete the following steps in turn:

- Click **Validate**
- Click **Commit** and **Yes** to confirm
- Click **Complete -> All tasks**, then click **Yes** to confirm
- Click **Activate**, then **Yes** to confirm

At this point, all of your definitions are moved to your J2EE Server and they are activated. You can restart your J2EE Server and start using your application.

Deploying CTGTesterECI

The installation of our ECI-based servlet application CTGTesterECI is as follows. It is not explained in great detail because it is very similar to the CTGTesterCCI installation described previously.

1. Start the AAT for z/OS (click **Start -> Programs -> IBM WebSphere for z/OS -> Application Assembly**). In the left pane, right-click **Applications** and select **Import**. Specify the location of the CTGTesterECI.ear file.
2. Right-click the **CTGTesterECI** under the Applications node and select **Validate**.
3. Right-click the **CTGTesterECI** under the Applications node and choose **Deploy**. Click **No** to the prompts to add required items. Look for the following message to confirm deployment has successfully completed:

```
BBO94009I Application CTGTesterECI was deployed.
```

4. Right-click the **CTGTesterCCI** application and select **Export**. Look for the following message to confirm deployment has successfully completed:

```
BBO94011I Application CTGTesterECI was exported to EAR file C:\Documents
and Settings\resident\aat\CTGTesterECI.ear.
```

System Management End-User Interface

The final task is to take the EAR file generated in the AAT for z/OS, and deploy it to a J2EE Server in WebSphere. Complete the following steps:

5. From the System Management End-User Interface (WebSphere Application Server for z/OS and OS/390 Administration), create a new conversation. Right-click **Conversations** and click **Selected -> Add**.
6. Right-click the J2EE Server where you installed the CICS ECI resource adapter and CTGTesterCCI and select **Install J2EE Application**.
7. In the EAR Filename field, enter the name of the EAR file you exported from the AAT for z/OS, then click **OK**.
8. You are required to provide a JNDI name for the Web application. This is used internally by WebSphere, and does not affect our application, so click the **Set Default JNDI Path & Name** button and then click **OK**.
9. You are now ready to activate the conversation. Right-click the conversation name (CTGTesterECI 27.6. 17:25) and complete the following steps in turn:

- Click **Validate**
- Click **Commit** and **Yes** to confirm
- Click **Complete -> All tasks**, then click **Yes** to confirm
- Click **Activate**, then **Yes** to confirm

10.3 Testing the configuration

We tested both our J2EE EJB application (CTGTesterCCI) and our servlet-based Web application (CTGTesterECI) in our J2EE Server, using the following two topologies:

- HTTP Transport Handler (port 80) -> J2EE Server
- IBM HTTP Server (port 99) -> WebSphere Plugin -> J2EE Server

> **Note:** This second topology was used to demonstrate how to authenticate the Web client and flow the security context onto CICS, since at the time of writing the HTTP Transport Handler did not support basic authentication. However, this support has now been made available with the PTF for APAR PQ59911, and we advise you to use this function if you require Web client authentication.

For further details on these different topologies, refer to Figure 10-2 on page 241.

How the HTTP request is matched to the definitions

Before testing, it is good to have a general understanding of how the URL is matched to your definitions. If something goes wrong, you then have an idea of where to look.

When using the IBM HTTP Server as the protocol catcher, the HTTP Server acts merely as a router for the HTTP request. The HTTP request is passed from the HTTP server into the WebSphere plugin and onto the Web container in the J2EE Server. Therefore, the HTTP Server can be used as a proxy to the J2EE Server or to handle security.

When using the HTTP Transport Handler in the J2EE Server, the HTTP request is received directly by the J2EE Server listening on the defined port, and the request is forwarded directly to the Web container in the J2EE Server.

Using the HTTP Server

The HTTP Server configuration is stored in the httpd.conf file. Within this file, there is one statement we are particularly interested in:

```
Service /CTGTester*/*
    /usr/lpp/WebSphere/WebServerPlugIn/bin/was400plugin.so:service_exit
```

The `Service` statement maps a request over to the WebSphere plugin environment. If the URL matches the given template (`/CTGTester*/*`), then the request is passed to the WebSphere plugin's executable module (was400plugin.so), which is defined in the second part of the URL, along with the entry point to be invoked in that module (service_exit).

Since there is no definition of our Web application (CTGTesterCCIWeb) in the WebSphere plugin configuration file (was.conf), then the HTTP request is automatically sent to the Web container in the J2EE Server, rather than executing in the WebSphere plugin within the HTTP Server address space.

Once the request is mapped over to the J2EE Server, the Web container will search through all the virtual-host/context-root pairs in its configuration file (webcontainer.conf) to see if the request matches. If the URL received matches a virtual-host/context-root pair, then the Web container knows to proceed. If no match is found, then WebSphere will reject the URL. We had the following definitions in our webcontainer.conf:

```
host.default_host.alias=wtsc66oe.itso.ibm.com:80,SC66:80,
                        wtsc66oe.itso.ibm.com:99
host.default_host.contextroots=/
```

This then is how our Web application functions:

1. The first request to our Web application using the URL http://wtsc66oe.itso.ibm.com:99/CTGTesterECIWeb/ is passed to the Web container in the J2EE Server.
2. The URI CTGTesterECIWeb causes the default welcome page to be loaded, which is defined in the Web deployment descriptor (web.xml) to be the Java Server Page (JSP) index.jsp.
3. This index.jsp contains an HTML form that is then used to invoke the servlet CTGTesterECIServlet.
4. WebSphere next goes looking for an application whose defined servletmapping value matches that implied on the received URL. In this example, the servletmapping string on the URL is CTGTesterECIServlet, and it knows this because the servletmapping value is whatever comes after the context root value on the URL. WebSphere looks through the web.xml files contained in each Web application's WAR file looking for a <url-pattern> tag that matches the servletmapping string on the URL. When it finds one, it takes the <servlet-name> value and goes to the point in the web.xml file where <servlet-class> is defined. This is the class file that is executed. In our web.xml (Example 10-4), the following servlet mapping was defined.

Example 10-4 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app id="WebApp">
   <display-name>CTGTesterECIWeb</display-name>
   <servlet>
      <servlet-name>CTGTesterECIServlet</servlet-name>
      <display-name>CTGTesterECIServlet</display-name>
```

```
<servlet-class>itso.cics.eci.testereci.CTGTesterECIServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>CTGTesterECIServlet</servlet-name>
      <url-pattern>CTGTesterECIServlet</url-pattern>
   </servlet-mapping>
   <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
   </welcome-file-list>
   <login-config>
      <auth-method>BASIC</auth-method>
      <realm-name>w3</realm-name>
   </login-config>
</web-app>
```

Using the HTTP Transport Handler

The processing of an HTTP request using the J2EE Server HTTP Transport Handler works in a very similar way to the IBM HTTP Server. However, since the request is received directly by the J2EE Server, no configuration of the HTTP Server is required. Thus the flow of control is as follows:

1. The request to the Web application using the URL `http://wtsc66oe.itso.ibm.com:80/CTGTesterECIWeb/` is received by the HTTP Transport Handler and passed to the Web container in the J2EE Server.
2. The URI `CTGTesterECIWeb` causes the default welcome page to be loaded, which is defined in the Web deployment descriptor (web.xml) to be the Java Server Page (JSP) index.jsp.
3. This index.jsp contains an HTML form that is then used to invoke the servlet CTGTesterECIServlet.
4. WebSphere next goes looking for an application whose defined servletmapping value matches that implied on the received URL and the servlet functions, as described in item 4 on page 267.

10.3.1 HTTP Transport Handler testing

We tested both our J2EE application CTGTesterCCI and our servlet-based application CTGTesterECI with our non-secure CICS region SCSCPJA1.

We deployed the applications as described in 10.2.5, “Deploying our sample enterprise applications” on page 258 and then used the following URLs to invoke the applications.

- `http://wtsc66oe.itso.ibm.com:80/CTGTesterECIWeb`

- `http://wtsc66oe.itso.ibm.com:80/CTGTesterCCIWeb`

Results with ECIRequest/servlet application

The initial HTLML page when running our CTGTesterECI Web application is shown in Figure 10-20.

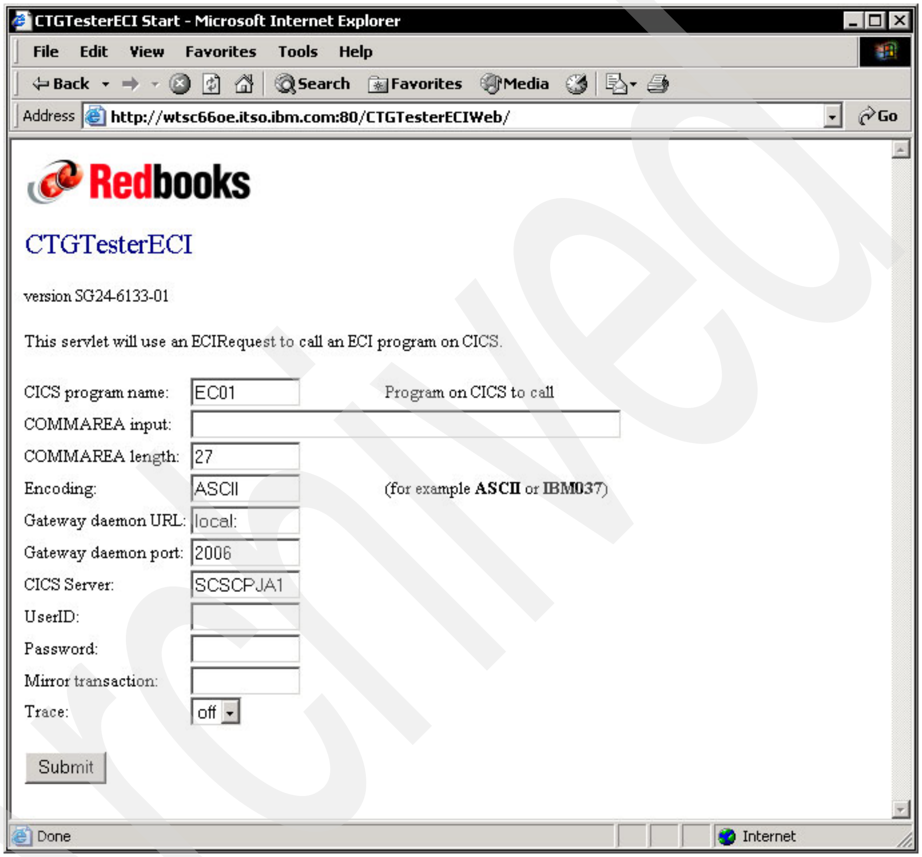


Figure 10-20 CTGTesterECI welcome page

We supplied the following input parameters:

CICS program name	ECIPROG2
COMMAREA length	35
Encoding	IBM037

Tip: We specified an Encoding of IBM037 (EBCDIC) since we wished to convert data within our servlet and not within CICS. If you wish to convert data within CICS, you will need to define a DFHCNV entry for each CICS program, and then you will need to specify ASCII on the HTML form. For further details on defining DFHCNV entries, refer to Appendix A, “DFHCNV and CICS data conversion” on page 329.

We then clicked the **Submit** button. The results of our successful call to the CICS program ECIPROG2 are shown in Figure 10-21. This displays the CICS APPLID, the date and time, and the user ID under which the request ran.

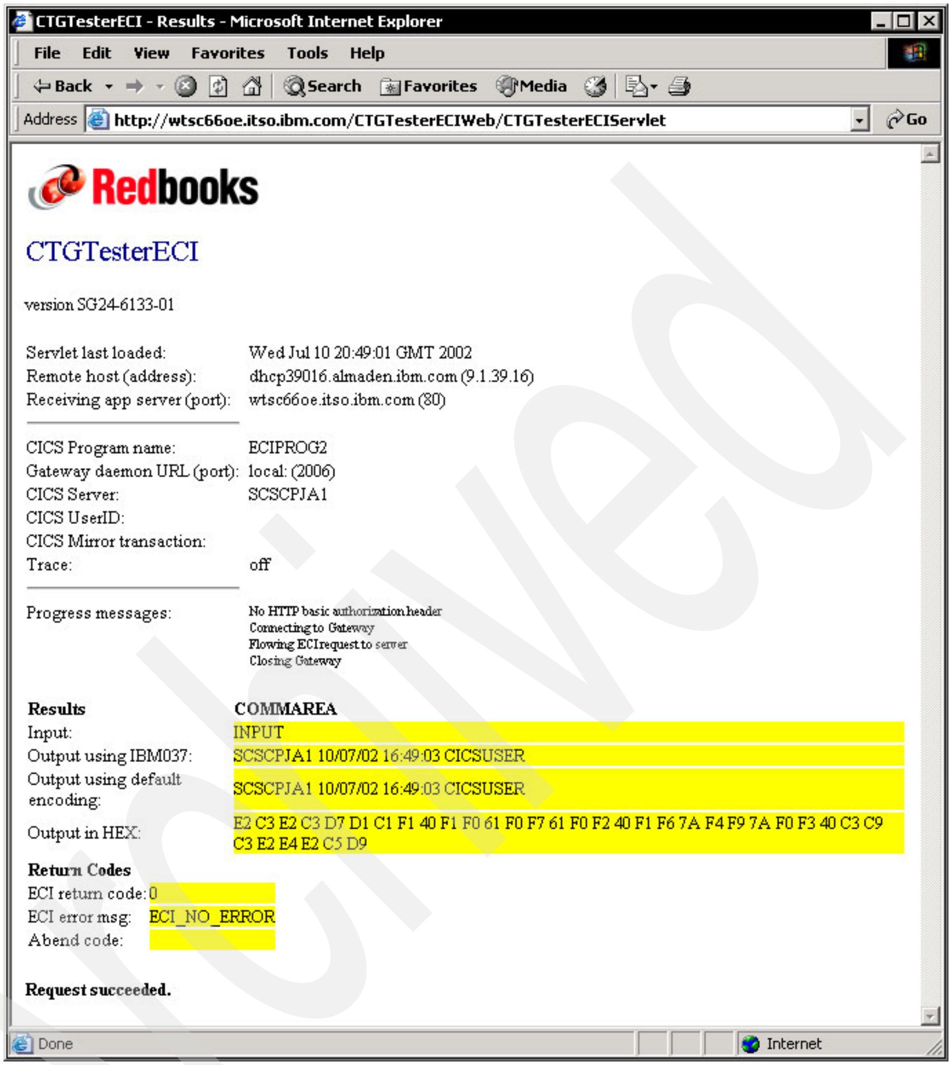


Figure 10-21 CTGTesterECI response page

The response from our call to our CTGTesterECI Web application (Figure 10-21) illustrates that the Web application successfully called the program ECIPROG2 in our CICS region SCSCPJA1. The COMMAREA output from ECIPROG2 shows that the request ran under default CICS user ID CICSUSER, since security was inactive in this CICS region.

CTGTesterCCI results

For our next test, we ran the J2EE application CTGTesterCCIWeb using the HTTP Transport Handler. The initial HTLML page is shown in Figure 10-22.

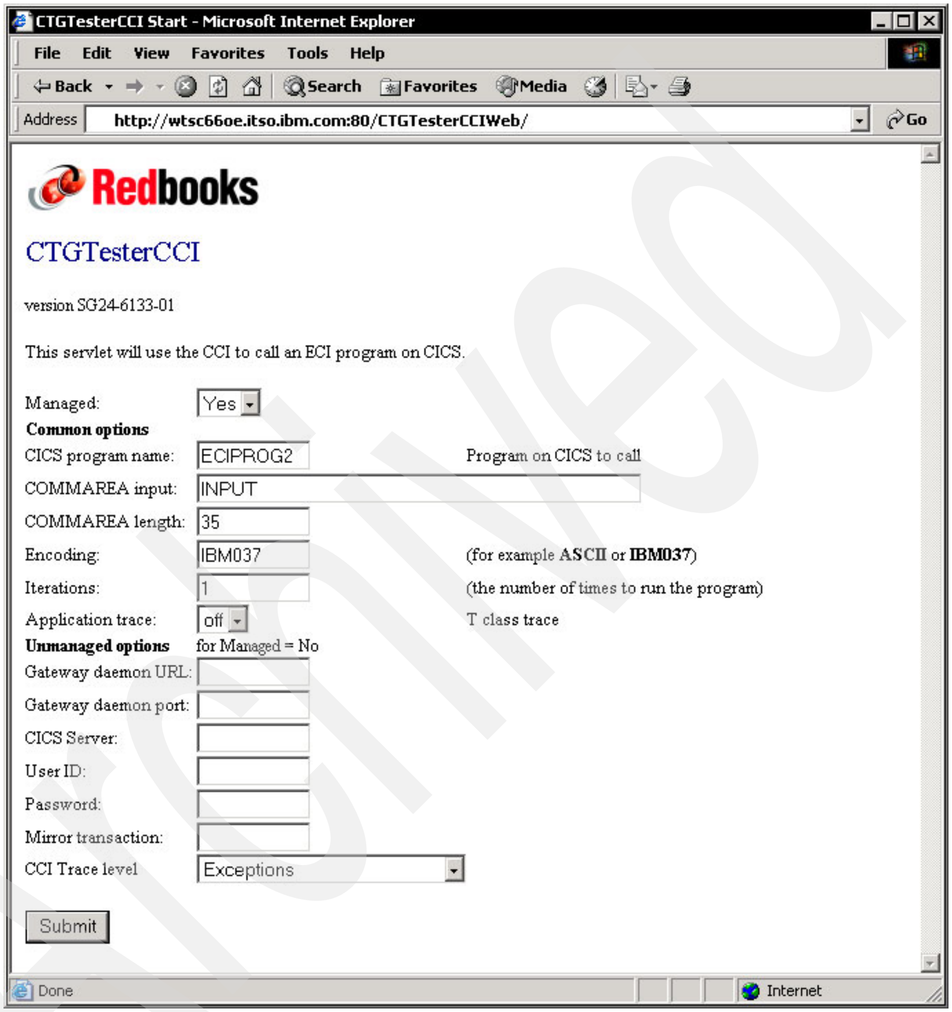


Figure 10-22 CTGTesterCCI welcome page

This time we are using a session bean in a managed environment to invoke our CICS region. Several of the options are now set on the managed connection factory, so it was necessary to modify the following parameters only:

CICS program name	ECIPROG2
COMMAREA length	35
Encoding	IBM037

We clicked **Submit** and received the output HTML form shown in Figure 10-23.

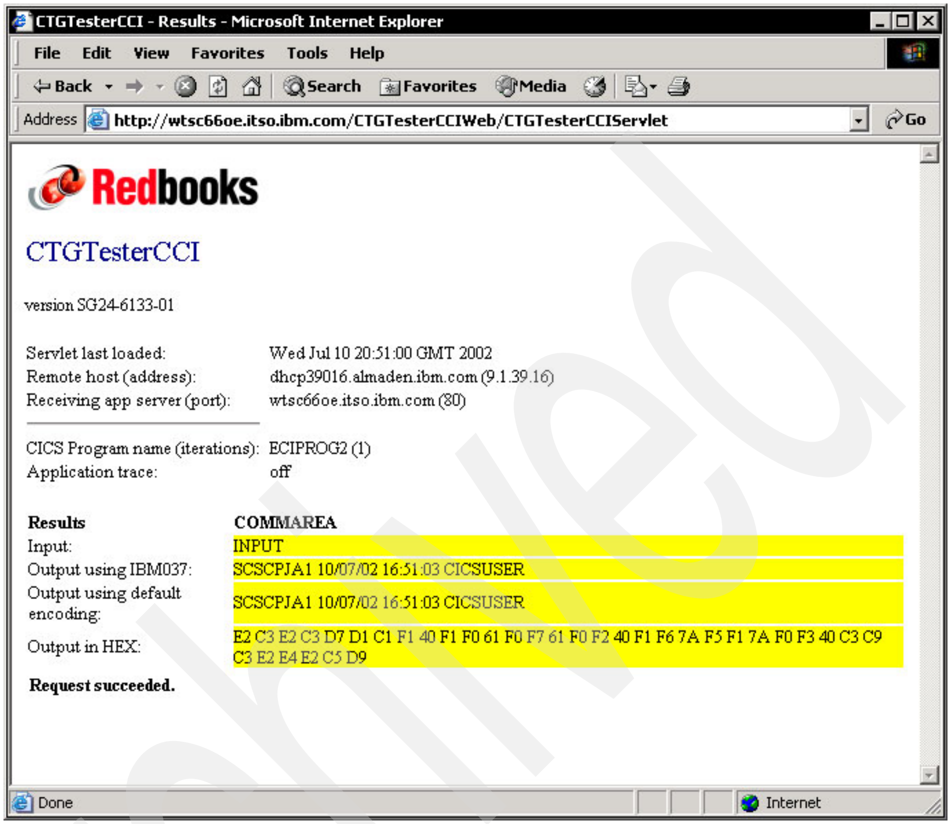


Figure 10-23 CTGTesterCCI response page

The response from our call to our CTGTesterCCI Web application (Figure 10-23) illustrates that the Web application successfully called the program ECIPROG2 in our CICS region SCSCPJA1, using our managed connection factory.

The COMMAREA output from ECIPROG2 shows that the request ran under default CICS user ID CICSUSER, since security was inactive in this CICS region.

10.3.2 HTTP Server and basic authentication testing

Next, we decided to route our requests to port 99 on which our HTTP Server was listening. To enable the HTTP Server to forward requests to the WebSphere plugin, we added the following service statement to httpd.conf:

```
Service       /CTGTester*/*
/usr/lpp/WebSphere/WebServerPlugIn/bin/was400plugin.so:service_exit
```

Since we did not have a matching `rooturi` in our WebSphere plugin configuration file (was.conf), the request was automatically routed over to the J2EE Server, where it was processed just like the previous requests handled by the HTTP Transport Handler.

We also modified our httpd.conf file to turn on turn on HTTP basic authentication for our Web applications. This meant our Web applications were now protected and the HTTP Server would allow access only if a valid user ID and password were supplied. The statements we added are as follows:

```
Protection CTGTesterWeb  {
                  ServerID        CTGTester
                  UserID          %%CLIENT%%
                  AuthType        Basic
                  PasswdFile      %%SAF%%
                  Mask            All
                  }
Protect /CTGTester*/* CTGTesterWeb
```

Results with ECIRequest/servlet application

We invoked our servlet application, CTGTesterECI Web application, using the following URL:

```
http://wtsc66oe.itso.ibm.com:99/CTGTesterECIweb
```

This caused the basic authentication challenge window to be presented to the Web client, as shown in Figure 10-24.

Figure 10-24 Basic authentication prompt

If you are running the HTTP Server with trace enabled, you will now see the output in Example 10-5 in the HTTP Server SYSOUT indicating the successful verification of the given user ID.

Example 10-5 HTTP Server SYSOUT

```
HTPasswd... user "CICSRS2" verified by SAF.
Authentic User... CICSRS2
Accepted.... by Mask (no ACL, only Protect)
............................................
APIClassExec Trying to match "/CTGTesterECIWeb/" with pattern
"/CTGTesterECIWeb/*".
APIClassExec using HTReqArgPath.
Pattern..... match SUCCEEDED.
```

Following the successful authentication of our user ID CICSRS2, we were presented with the CTGTesterECI welcome page shown in Figure 10-25.

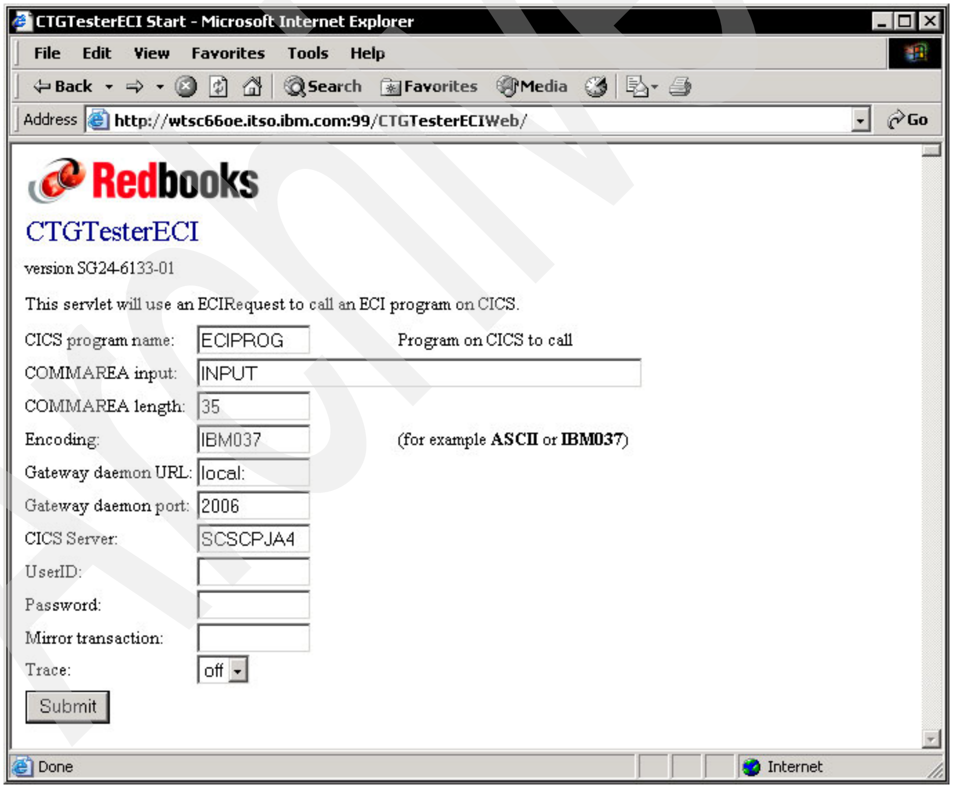


Figure 10-25 CTGTesterECI welcome page

We changed the following parameters:

CICS program name	ECIPROG2
COMMAREA length	35
Encoding	IBM037
CICS Server	SCSCPJA4

We then clicked **Submit**, which caused the ECI request to be flowed to our CICS region using the local CICS TG, and the successful results were displayed as shown in Figure 10-26.

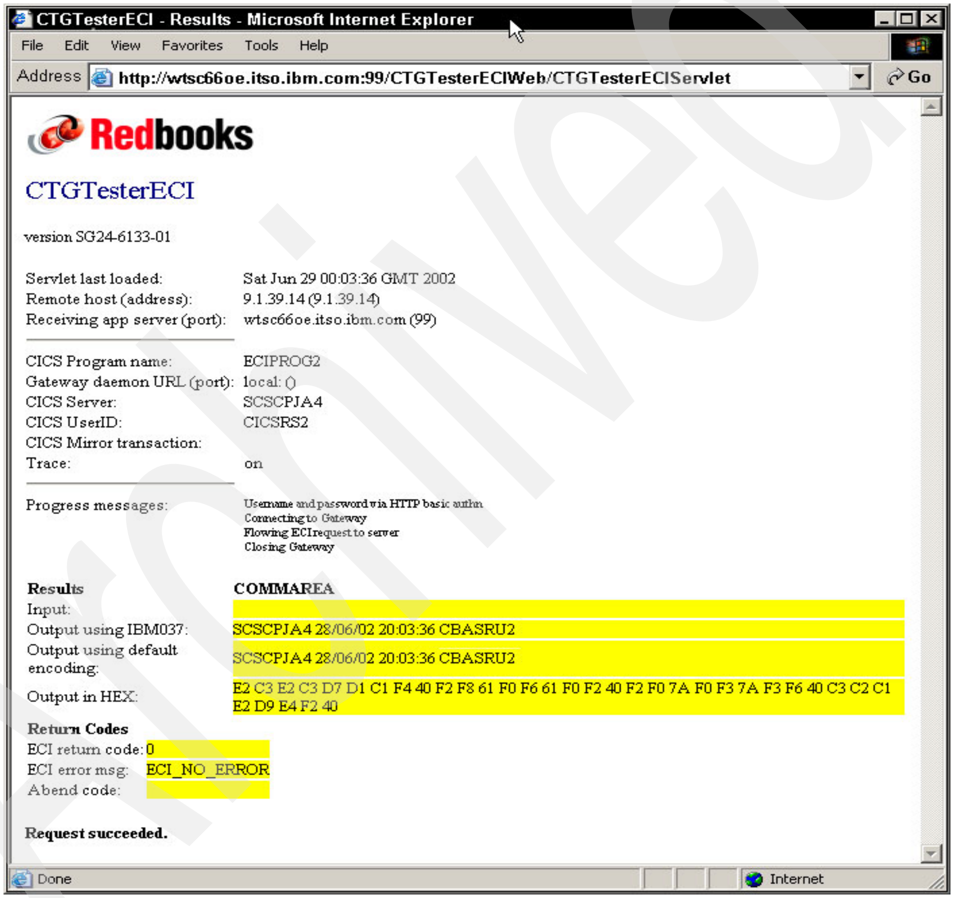


Figure 10-26 CTGTesterECI response page

The output from ECIPROG2 shows that the request ran successfully in our secure CICS region SCSCPJA4, and ran under the user ID CBASRU2. This is the user ID of the J2EE Server, and *not* the client user ID authenticated by the HTTP Server.

Once the request had successfully executed in CICS we checked the CICS MSGUSR log which shows useful information about the link user ID used.

Example 10-6 CICS log MSGUSR

```
DFHSN1400 07/10/2002 19:14:11 SCSCPJA4 Session signon for session WC1 by user
CBASRU2 is complete.
DFHSN1500 07/10/2002 19:14:11 SCSCPJA4 Session signoff for session WC1 is
complete. 1 transactions entered with  0  errors.
```

The link user ID is an additional level of security used for MRO and ISC requests. For EXCI or MRO requests, the link user ID defaults to the user ID under which the connected region runs. In our case, CBASRU2 is the user ID under which the J2EE Server started task runs. This user ID must have access to all the same resources as the flowed user ID (CICSRS2). To disable link security, it is possible to make the systems *equivalent* by setting the USERID parameter in the EXCI SESSIONS definition. For more details, refer to “Link security” on page 107.

Note: Our tests showed that in WebSphere V4 the client user ID is no longer automatically propagated through to CICS when using the ECIRequest class from a servlet, as is the case with servlets that run in the WebSphere V3.5 plugin. This is because in the WebSphere V4 J2EE Server the RACF ACEE of the thread is not switched to the user ID of the client. This behavior is not affected by modifying the ThreadID parameter in the EJB deployment descriptor (see “Security settings” on page 280) since this parameter affects only the EJB container and not the Web container, which is where servlets are executed.

Thus if you require propagation of the user ID, we suggest you either use J2EE applications or modify your servlet code to extract the user ID from the HTTP basic authentication header, and copy it into the `strUserid` parameter in the `ECIRequest` object. Sample code to do this is provided in our CTGTesterECI application.

Results with J2EE application

To install our J2EE application, CTGTesterCCI, we first started a new conversation using the WebSphere for z/OS Administration application and defined a new CICS ECI resource adapter for our secure CICS region SCSCPJA4 (for further details refer to “Installation of the CICS ECI resource adapter” on page 252). We then deployed our J2EE application in order for it to use this new resource adapter. This process is explained in detail in “Installing the J2EE application” on page 261.

When CTGTester was successfully re-installed, we entered the following URL and received the basic authentication security pop-up window (Figure 10-27).

```
http://wtsc66oe.itso.ibm.com:99/CTGTesterCCIWeb
```

Figure 10-27 Basic authentication prompt

We entered our user ID (CICSRS2) and password and received the index.jsp file as shown in Figure 10-28.

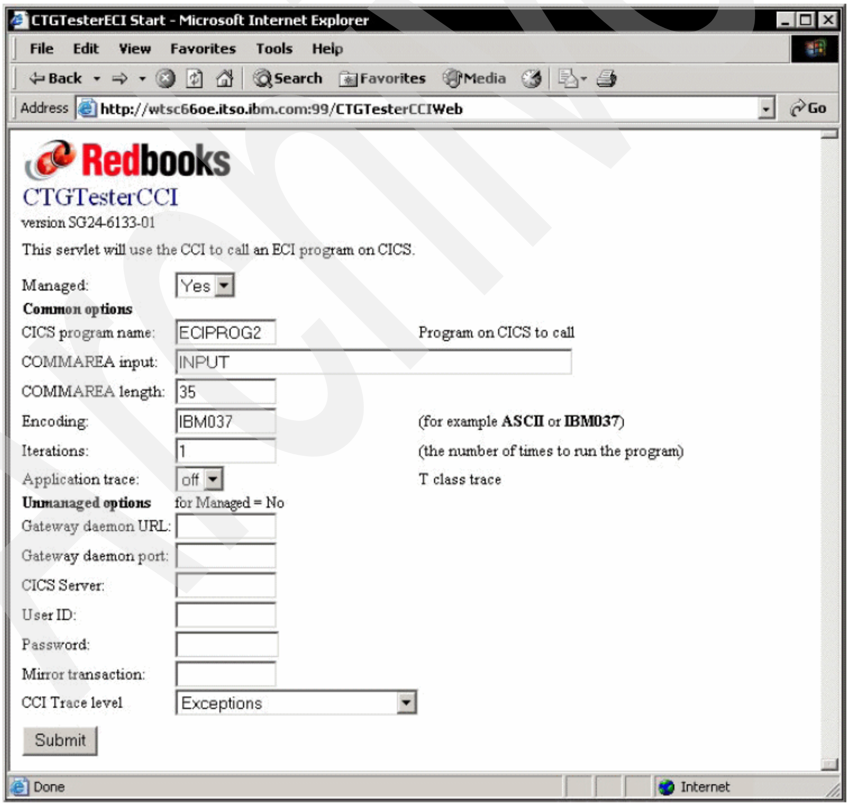


Figure 10-28 CTGTesterCCI welcome page

Since we are using a session bean in a managed environment, several of the options are now set on the managed connection factory, so it was necessary to modify only the following parameters:

CICS program name	ECIPROG2
COMMAREA length	35
Encoding	IBM037

We clicked **Submit** and received the output HTML form shown in Figure 10-29.

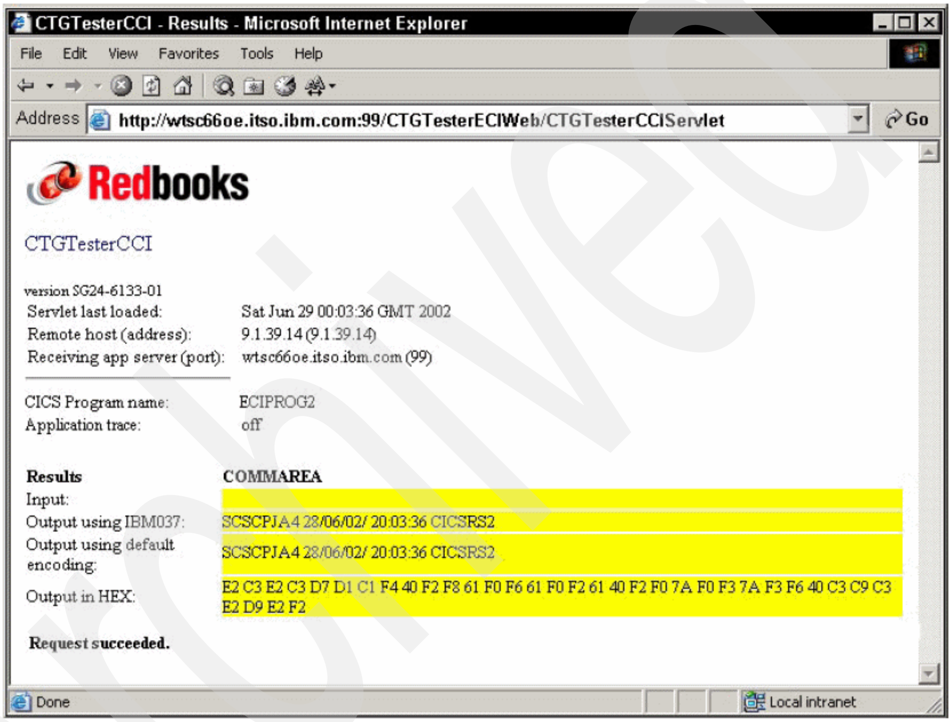


Figure 10-29 CTGTesterCCI response page

The output from ECIPROG2 shows that the request ran successfully in our secure CICS region SCSCPJA4, and this time ran under the flowed user ID CICSRS2. This is the client user ID authenticated by the HTTP Server using the `%%CLIENT%%` setting in the protection directive. The message DFHSN1400 in the CICS MSGUSER log also showed that the link user ID was again CBASRU2, the user ID of the J2EE Server region.

> **Note:** Our J2EE application used a managed Connection Factory. However, with a J2EE application it is also possible to create a non-managed application, which does not use the JNDI to look up a pre-defined Connection Factory. In this case the user ID flowed to CICS will be the J2EE Server's user ID, and not the client user ID. The only way to automatically propagate the client security context in this case is to use the ThreadID parameter as discussed in the following text.

Security settings

In WebSphere V4 for z/OS, many different security settings can affect the user ID that is flowed to CICS. We performed further tests to investigate how the following parameters affected the flowed user ID and link user ID for both a servlet/ECI application and a J2EE/CCI application. Our results are summarized below:

- %%CLIENT%%

  This is set in the HTTP Server Protection directive. It controls the user of basic authentication by the HTTP Server, and can be used together with the RunAS parameter in the EJB deployment descriptor to control the user ID flowed to CICS.

- Surrogate

  This parameter is set in the WebAuth.UnauthenticatedUserSurrogate parameter in the J2EE Server webcontainer.conf file. It is used to specify the user ID for unauthenticated requests. We found that if this parameter is set, it will cause the client user ID set by the HTTP Server using basic authentication to *not* be used for an EXCI request to CICS.

> **Important:** If you wish to use basic authentication with J2EE applications, it is recommended that you define security in the Web application's deployment descriptor, using the <security-constraint> and <login-config> settings. This will allow the Web container to authenticate your requests. The function for this is provided in APAR PQ59911 and PQ55181, and is not covered in this redbook. For further details, refer to the redbook *z/OS WebSphere and J2EE Security Handbook*, SG24-6846.

- RunAS

  This parameter is set on the +RunAS tab in the EJB deployment descriptor and can be set for each of the methods in the home and remote interfaces of a session bean. It can have the value *Caller, Server,* or *Role.* We used the default value *Caller*, which means the user ID of the caller is propagated to the J2EE resource (the ECI resource adapter). Using *Server* or *Role* would cause either the Server's user ID or EJB role security to be used.

- ThreadID

  This parameter is set on the EJB deployment descriptor in the +ThreadID tab by selecting **Set OS Thread Identity to RunAS Identity** for the methods in the home and remote interface. The default is for the value not to be set. It controls the operating system identity on the thread of execution and allows the security context of the thread to be switched to that of the caller itself. If set for a J2EE/EJB application using the ECI resource adapter, this will change the link user ID for the EXCI request to be the flowed user ID rather than the user ID of the J2EE Server.

- Res-Auth

  This parameter controls whether the container or application component will determine the user ID. This defaults to *container* and this is what we used, since this will honor the settings for ThreadID and RunAS. If Res-Auth is set to *application* then the user ID and password are taken from the values specified in the `getConnection()` method.

10.4 Problem determination

In this section, we document useful information we learned while configuring this scenario and further information on problem determination and tracing.

10.4.1 Tips and utilities

In this section, you will find commands and utilities to use in debugging problems with your configuration. You will also find some additional information about topics discussed in this chapter.

WebSphere logs

WebSphere Application Server uses the MVS Logger to store its log data, and therefore for setup you need to define policies to the MVS Logger for this purpose. In order to access and read this log information, you need to have access to a sample CLIST called BBORBLOG, which is supplied on the SBBOEXEC data set, which can be run using the following command:

```
EXEC 'BBO66.SBBOEXEC(BBORBLOG)'    'WAS.SC66.ERROR.LOG'
```

Where `WAS.SC66.ERROR.LOG` is the name of the MVS Logger logstream definition containing the log data. A sample of the type of information you might find when you enter this command is shown in Example 10-7.

Example 10-7 Log data from MVS Logger

```
2002/07/22 12:26:26.991 01 SYSTEM=SC66 SERVER=C1OASR2A JobName=C1OASR2S
ASID=0X03FE PID=0X050E0061 TID=0X10E4E208 00000000 c=UNK ./bbolsys.cpp+882 ...
BBOU0632E JVM EXIT API DRIVEN. JVM EXITING WITH CODE=
2002/07/22 11:48:45.951 01 SYSTEM=SC66 SERVER=C1SMGT01 JobName=C1OSMSS
ASID=0X03F2 PID=0X000E00A4 TID=0X10E26C80 00000000 c=UNK ./bbolsys.cpp+882 ...
BBOU0632E JVM EXIT API DRIVEN. JVM EXITING WITH CODE=
```

Common errors

We found that if we included all the required JAR files as stated by the AAT for z/OS, this could lead to class loader problems. To avoid this problem, we did not import the stated classes, and instead specified them on the J2EE Server CLASSPATH setting.

Useful commands

For problem determination with your LDAP server, you will need a browser capable of browsing the JNDI name space. For details of the *Softerra LDAP browser* we used with our LDAP server, refer to Chapter 4, “Configuring the JNDI server” in the IBM redbook, *Enterprise JavaBeans for z/OS and OS/390, CICS Transaction Server V2.2*, SG24-6284.

Also for more advanced investigation of the JNDI name space when using an LDAP server, you can use the LDAP command-line utilities:

- ldapsearch
- ldapmodify
- ldapadd
- ldapdelete
- ldapmoddrn

10.4.2 Tracing

Various levels of CICS TG tracing can be enabled from WebSphere. They are as follows:

- J2EE resource adapter trace
- CICS TG Java client trace
- CICS TG JNI trace

J2EE resource adapter trace

Resource adapter tracing is used to control tracing of the flow of execution through the CCI classes. When using a non-managed connection factory, it is activated programmatically by invoking the `setTraceLevel()` method on the `ECIManagedConectionFactory` object. Our sample application CTGTesterCCI provides the option to activate J2EE resource adapter tracing in this way.

When using a managed environment, J2EE resource adapter tracing is instead activated by using the Administration application as described in the following section.

Using the Administration application, set the Trace Level property in the CICS_ECIConnectionFactory J2EE Resource to one of the following values:

0 Disable all tracing
1 Output exception trace stacks
2 Output method entry and exit stack traces
3 Output debug trace entries

We set this value to 2 as shown in Figure 10-30 on page 283, and we also set the LogWriter Recording parameter to `enable`.

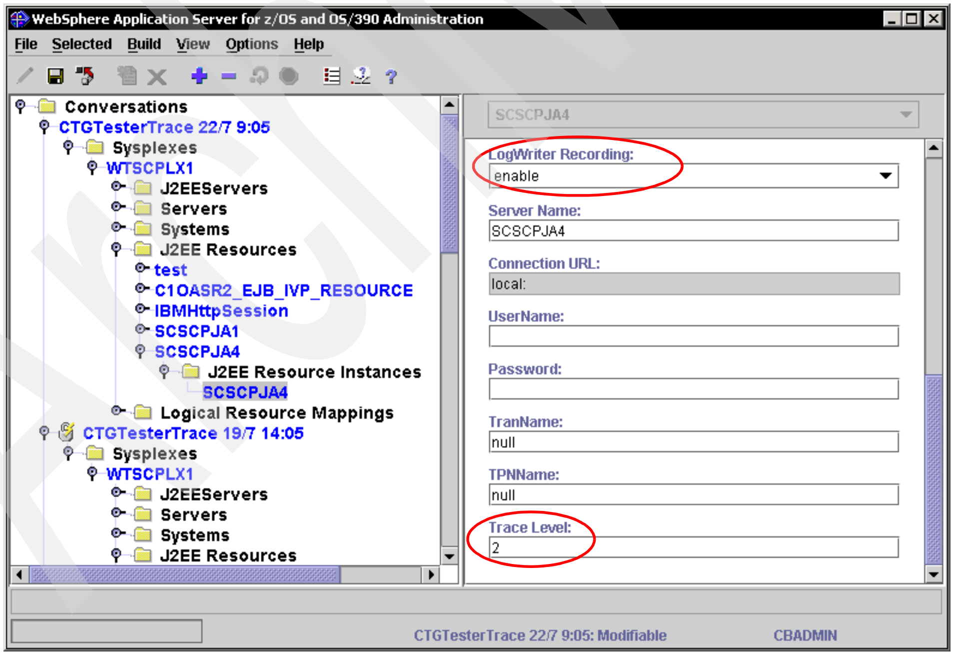


Figure 10-30 Trace Level settings

We then added the following to our J2EE Server jvm.properties file:

```
com.ibm.ws390.trace.settings=/WebSphereC1/CB390/controlinfo/envfile/
WTSCPLX1/C1OASR2A/trace.dat
```

This line names a trace.dat file that we created in the same directory as the jvm.properties file. In the trace.dat file we added the following two statements:

```
com.ibm.ws.classloader.*=all=enabled
com.ibm.ws390.logwriter.*=all=enabled
```

We then restarted our J2EE Server using the Operations application and trace was now active, in the J2EE Server, and written to the SYSPRINT of the J2EE Server. An example of the trace output is shown in Figure 10-8.

Example 10-8 J2EE trace output

```
Trace: 2002/07/22 18:24:05.074 01 t=7D20B8 c=8.2 key=P8 (13007002)
 FunctionName: com.ibm.ws390.logwriter.SCSCPJA4
 SourceId: com.ibm.ws390.logwriter.SCSCPJA4
 Category: DEBUG
  ExtendedMessage: 18:24:05:073 : ÝWebSphere t=007d20b8:2cf5c220¨ :  -->
com.ibm.connector2.cics.ECIManagedConnectionFactory.cre
onnectionFactory()
parms=com.ibm.ws390.connmgmt.WS390AppServerConnectionManager@36286ca5
Trace: 2002/07/22 18:24:05.074 01 t=7D20B8 c=8.2 key=P8 (13007002)
 FunctionName: com.ibm.ws390.logwriter.SCSCPJA4
 SourceId: com.ibm.ws390.logwriter.SCSCPJA4
 Category: DEBUG
  ExtendedMessage: 18:24:05:074 : ÝWebSphere t=007d20b8:37bbeca5¨ :  -->
com.ibm.connector2.cics.ECIConnectionFactory.super() pa
ConnectionManager=Ýcom.ibm.ws390.connmgmt.WS390AppServerConnectionManager@36286
ca5¨  ManagedConnectionFactory=Ýcom.ibm.connector
cs.ECIManagedConnectionFactory@2cf5c220 ConnectionURL="local:"
ServerName="SCSCPJA4" PortNumber=2006 Userid="null"" TranName=nul
NName=null¨
Trace: 2002/07/22 18:24:05.075 01 t=7D20B8 c=8.2 key=P8 (13007002)
 FunctionName: com.ibm.ws390.logwriter.SCSCPJA4
 SourceId: com.ibm.ws390.logwriter.SCSCPJA4
 Category: DEBUG
  ExtendedMessage: 18:24:05:075 : ÝWebSphere t=007d20b8:37bbeca5¨ :  <--
com.ibm.connector2.cics.ECIConnectionFactory.super()
```

Tip: We found that J2EE resource adapter tracing did not provide much useful information for debugging purposes. Instead if you wish to enable tracing we suggest you combine J2EE resource adapter tracing with CICS TG Java client trace (T class). Using both of these traces will show the end-to-end flows from the J2EE resource adapter to the `JavaGateway` methods.

For further details on tracing J2EE resources in CICS, refer to Chapter 4 in *WebSphere Application Server V4.0.1 for z/OS and OS/390, Messages and Diagnosis*, GA22-7837.

Java client trace

CICS TG Java client tracing can be set programmatically within the servlet by adding the following statements to the Java application:

```
com.ibm.ctg.client.T.setDebugOn(true);
com.ibm.ctg.client.T.setTimingOn(true);
```

In our CTGTesterECI Web application the Trace option on the index.jsp welcome page controls this tracing. Output from this trace will go to the SYSPRINT log of the J2EE Server, since this is what we set up for the TRACEBUFLOC, as shown on page 286.

Note: Enabling CICS TG Java client tracing in an application will enable it for all applications running in the application server, as a consequence of the static nature of the T class.

CICS TG JNI trace

CICS TG JNI trace is output from the Gateway JNI module (libCTGJNI.so) that acts as the interface between the Gateway and the underlying native transport (EXCI). We found it to be a very useful means of problem determination.

It is controlled with the Java system property `gateway.T.setJNITFile`. The property value specifies the file name where the JNI trace is output. This must be set in the JVM properties file for the J2EE Server. To do this, we performed the following steps:

1. Edited our J2EE Server instance's jvm.properties file, located in the directory /WebSphereC1/CB390/controlinfo/envfile/WTSCPLX1/C1OASR2A.
2. Added the following Java property:

   ```
   gateway.T.setJNITFile=/tmp/wasctgjni.trc
   ```

3. Restarted our J2EE Server instance using the Operations application.

You can see the output in our JNI trace file (/tmp/wasctgjni.trc) for a simple request to our CTGTesterCCI Web application in Example 10-9.

Example 10-9 CICS TG JNI trace output from WebSphere

```
CICS Transaction Gateway JNI Trace file for z/OS Version 5.0, Build Level
c000-20020621
20:13:59.451 [020e008c,10e616a0,WS007208    ] : CCL6813I CcicsInit: Running
under Websphere z/OS.
```

```
20:13:59.451 [020e008c,10e616a0,WS007208    ] : CCL6880I Specific pipe used.
NETNAME=WebSphere t=007d20b8
20:13:59.622 [020e008c,10e616a0,WS007208    ] : CCL6800I ECI parameters on
entry.  Call_Type = 1, Extend_Mode = 0, Luw_Token = 0, C
ommarea_Length=35,Cics_Rc=0,Flags=2,outbound Commarea length=2.
20:13:59.622 [020e008c,10e616a0,WS007208    ] : CCL6801I CcicsECI: Performed
parameter conversion.  parameter name = jstrServer, pa
rameter value = SCSCPJA4.
20:13:59.623 [020e008c,10e616a0,WS007208    ] : CCL6801I CcicsECI: Performed
parameter conversion.  parameter name = jstrUserid, pa
rameter value = CBASRU2 .
20:13:59.623 [020e008c,10e616a0,WS007208    ] : CCL6801I CcicsECI: Performed
parameter conversion.  parameter name = jstrProgram, p
arameter value = ECIPROG2.
20:13:59.624 [020e008c,10e616a0,WS007208    ] : CCL6802I CcicsECI: Input
commarea information.  commarea length = 35, commarea addr
ess = 1e7c2728.
20:13:59.624 [020e008c,10e616a0,7208CCL6802IC : CCL6810I connopen: First
request on TCB address = 7d20b8.
20:13:59.628 [020e008c,10e616a0,WS007208    ] : CCL6804I CcicsECI: Returned
commarea information.  commarea length = 35, commarea a
ddress = 1e7c2728, inbound commarea data length =35.
20:13:59.629 [020e008c,10e616a0,WS007208    ] : CCL6805I ECI parameters on
exit.  Call_Type = 1, Extend_Mode = 0, Luw_Token = 0, Co
mmarea_Length = 35, Cics_Rc = 0, AV = 2.
```

Tip: New in CICS TG for z/OS V5.0 is the default logging of EXCI return codes. These are written to unique files in the directory $HOME/ibm/ctgjnilog.* and should be checked before resorting to a CICS TG JNI trace.

J2EE Server tracing

To enable tracing in the WebSphere Application Server J2EE Server, you must place a new entry in the environment variable list for the appropriate J2EE Server. This can be done either from within WebSphere Application Server for z/OS administration console, or simply by editing the current.env file.

Our current.env file was located in the following directory:

/WebSphereC1/CB390/controlinfo/envfile/WTSCPLX1/C1OASR2A/current.env

To activate JVM tracing, we added the following variables to this file:

- `JVM_LOGFILE=/tmp/WAS.jvm.log`
- `JVM_DEBUG=1`

This caused JVM trace entries to be written out to the HFS file /tmp/WAS.jvm.log.

To activate J2EE Server tracing, we also added the following variables:

- `TRACEALL=1`
- `TRACEBUFFLOC=SYSPRINT`

This causes level 1 trace entries to be written to the SYSPRINT of the J2EE Server.

> **Note:** Tracing in the J2EE Server takes effect as soon as the server is restarted. Since this means that the startup process itself will be traced, you will notice that startup will take a longer time to complete than normal.

You may have noticed the operand we chose to open the variable list for input was JVM_LOGFILE. This operand indicates where output from the trace will be sent. The value we specified was `/tmp/WAS.jvm.log`. A sample view of the trace data sent to this file is shown in Example 10-10.

Example 10-10 Sample output from JVM trace

```
Opened /usr/lpp/java/IBM/J1.3/lib/rt.jar in 134 ms¨
Opened /usr/lpp/java/IBM/J1.3/lib/i18n.jar in 6 ms¨
Loaded java.lang.NoClassDefFoundError from /usr/lpp/java/IBM/J1.3/lib/rt.jar¨
Loaded java.lang.Class from /usr/lpp/java/IBM/J1.3/lib/rt.jar¨
Loaded java.lang.Object from /usr/lpp/java/IBM/J1.3/lib/rt.jar¨
Loading superclasses of java/lang/Object¨
Loaded java.lang.Throwable from /usr/lpp/java/IBM/J1.3/lib/rt.jar¨
Loading superclasses of java/lang/Throwable¨
Loaded java.io.Serializable from /usr/lpp/java/IBM/J1.3/lib/rt.jar¨
Loading superclasses of java/io/Serializable¨
Preparing java/lang/Object¨
Initializing java/lang/Object¨
```

HTTP Server tracing

If you are using the HTTP Server as the HTTP protocol catcher, trace can be activated by setting the `-v` or `-vv` parameters in the startup procedure. To activate trace in a running server, you can use the SDSF command:

```
F SCSWEBC6,APPL=-VV
```

11

CICS TG and WebSphere Application Server for Windows

In this chapter, we describe how we configured WebSphere Application Server Advanced Edition (WebSphere AE) for Windows to use the CICS Transaction Gateway (CICS TG) Java classes to make External Call Interface (ECI) calls to our CICS Transaction Server (CICS TS) V2.2 region. See Figure 11-1 on page 290.

We used two test applications; CTGTesterECI and CTGTesterCCI. CTGTesterECI uses a servlet and the CICS TG Java classes, CTGTesterCCI uses a session bean and the J2EE resource adapter. We first tested using local mode to connect directly to the Client daemon on the WebSphere machine, then we tested using remote mode to connect to a CICS TG on z/OS.

11.1 Preparation

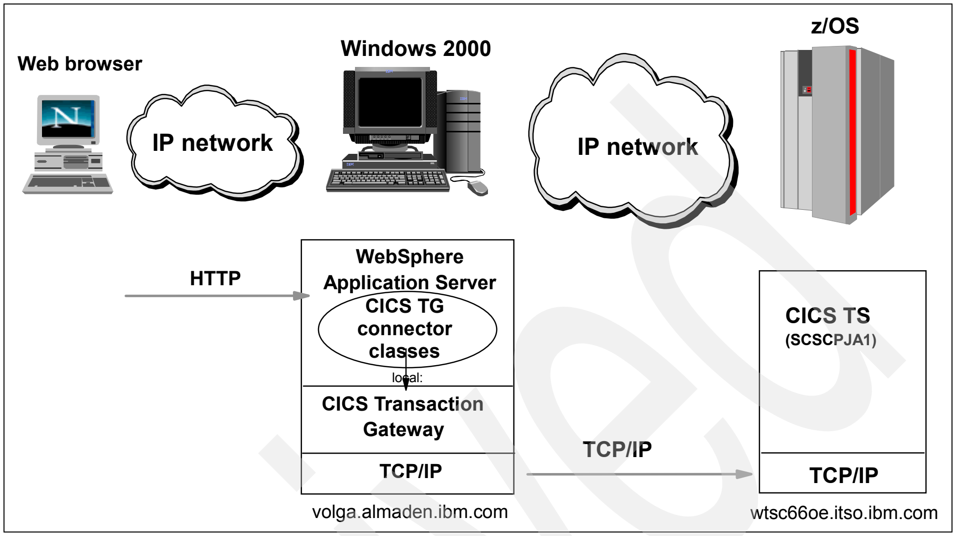


Figure 11-1 Software components: CICS TG and WebSphere for Windows scenario

In the following sections, we detail how we configured WebSphere Application Server to work in conjunction with the CICS TG, and how we then tested our configuration using our sample applications, CTGTesterCCI and CTGTesterECI. For further details on how we developed these applications, refer to Appendix B, “Sample applications” on page 337.

The CTGTesterECI enterprise application is a servlet-based ECI application that calls a CICS program and displays the results. Figure 11-2 illustrates the complete architecture of the application.

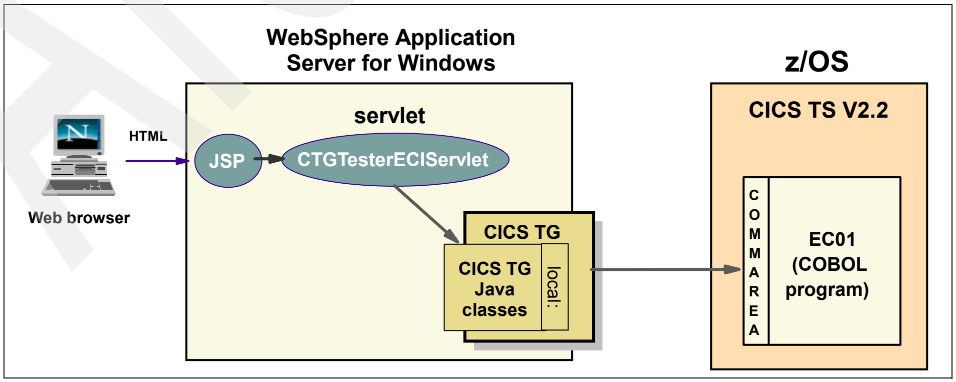


Figure 11-2 Application architecture of CTGTesterECI, local mode

The flow of control through the enterprise application is as follows:

- The JavaServer Page (JSP) index.jsp contains an HTML form that starts the servlet CTGTesterECIServlet. Several parameters are sent to the servlet: the name of the CICS program to call, the encoding to use, and details of the Gateway daemon, among others.
- The servlet makes the call to CICS using the `ECIRequest` class in the CICS TG Java class library. Once the call has completed, the servlet forwards the results to one of three JSPs, depending on whether the request was successful, an error occurred, or an exception occurred.
- The JSP formats and displays the results of the request.

This enterprise application is stored in the file CTGTesterECI.ear. To obtain this file, see Appendix C, "Additional material" on page 389.

The CTGTesterCCI enterprise application calls a CICS program using the CICS ECI resource adapter and displays the result. Figure 11-3 illustrates the complete architecture of the enterprise application.

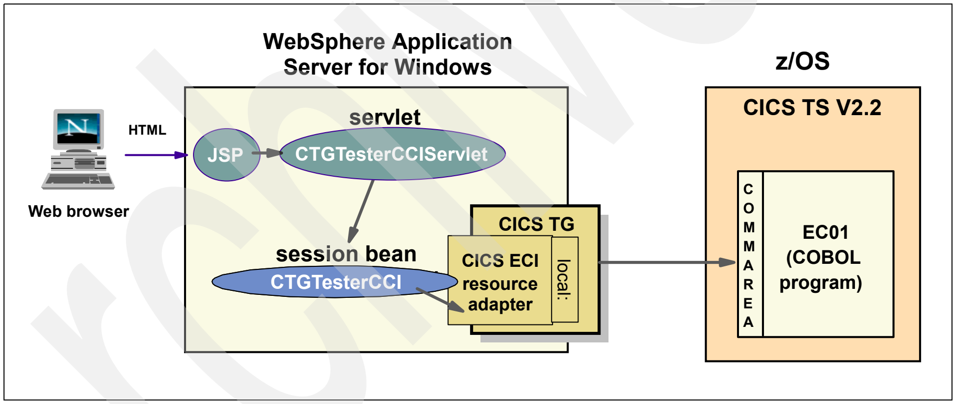


Figure 11-3 Application architecture of CTGTesterCCI, local mode

The enterprise application consists of the following steps:

- The JSP page index.jsp contains an HTML form that starts the servlet CTGTesterCCIServlet. Several parameters are sent to the servlet: the name of the CICS program to call, the COMMAREA input and length, and the encoding to use.
- The servlet passes these parameters on to the CTGTesterCCI session bean.
- The session bean makes the call to CICS using the CICS ECI resource adapter, and returns the response to the servlet.

- The servlet forwards the results to one of two JSPs, depending on whether the request was successful or not.
- The JSP formats and displays the results of the request.

The servlet also allows the use of an unmanaged connection, in which case several additional parameters, such as CICS server and user details, may be specified.

This enterprise application is stored in the file CTGTesterCCI.ear. To obtain this file, see Appendix C, “Additional material” on page 389.

For details on how we configured the underlying connectivity to our CICS Transaction Server region using the TCP/IP protocol, refer to Chapter 5, “TCP/IP connections to CICS” on page 81.

11.1.1 Software checklist

We used the levels of software described in Table 11-1.

Table 11-1 Software checklist: WebSphere for Windows

Windows workstation	**z/OS**
Windows 2000 Service Pack 2	z/OS V2.1
IBM DB2® Universal Database™ Enterprise Edition 7.2.	CICS Transaction Server V2.2
IBM HTTP Server V1.3.19 (included with WebSphere AE)	
WebSphere Application Server V4.03 for Windows - Advanced Edition	
CICS Transaction Gateway for Windows V5.0	CICS Transaction Gateway for z/OS V5.0
Microsoft Internet Explorer V6.0.26	

Note: We used WebSphere Application Server V4.03 Advanced Edition because it comes with support for J2EE Connector Architecture V1.0. WebSphere AE V4.01 does not, and requires the Connector Architecture for WebSphere Application Server (technology preview) to be installed on top of WebSphere. Additionally, the CICS TG recommends the use of WebSphere V4.03 for considerably better J2EE performance than WebSphere V4.02.

11.1.2 Definitions checklist

Before you configure the products, we recommend that you acquire definitions for the following parameters. The values shown in Table 11-2 are the definitions we used on our Windows workstation.

Table 11-2 Definitions checklist: WebSphere for Windows

Property	Value
IBM HTTP Server install directory	C:\IBM HTTP Server
WebSphere install directory	C:\WebSphere\AppServer
DB2 install directory	C:\Program Files\SQLLIB
CICS TG install directory	C:\Program Files\IBM\IBM CICS Transaction Gateway
Windows 2000 workstation hostname	volga
CICS Server name	SCSCPJA1

In addition we also had to deploy our CICS COBOL application EC01, and to configure a DFHCNV data conversion template in CICS for use by this program. Refer to Appendix A, “DFHCNV and CICS data conversion” on page 329 for more details.

11.1.3 Installing WebSphere Application Server

We first installed IBM DB2 7.2. We selected a typical install and took note of the DB2 password. We chose not to install the OLAP starter kit. Once DB2 was installed, we upgraded the JDBC level to V2.

We then installed WebSphere Application Server using the supplied setup.exe application. During the installation we selected the Typical Installation. We supplied the DB2 password when asked. When the installation was completed, we rebooted the Windows 2000 machine.

For further information and guidance concerning installing WebSphere, refer to the Redbook *IBM WebSphere V4.0 Advanced Edition Handbook*, SG24-6176.

11.2 Configuration

In this section, we discuss the configuration of our WebSphere Application Server on Windows 2000. We detail the following steps:

- Starting the administrative console
- Installing the CICS ECI resource adapter
- Deploying the CTGTesterECI application
- Deploying the CTGTesterCCI application

The CTGTesterCCI application makes use of the CICS ECI resource adapter, so it must be installed for CTGTesterCCI to work. CTGTesterECI does not need the CICS ECI resource adapter installed.

11.2.1 Starting the administrative console

Before starting the WebSphere administrative console, we started the Admin Server. This was done by selecting **Start -> Programs -> IBM WebSphere -> Application Server V4.0 AE -> Start Admin Server**, but we could have also started it using the IBM WS AdminServer 4.0 Windows service.

Once the Admin Server has started, we started the WebSphere administrative console by selecting **Start -> Programs -> IBM WebSphere -> Application Server V4.0 AE -> Administrator's Console.** A pop-up window was displayed stating that the console was loading, and then the window shown in Figure 11-4 was displayed.

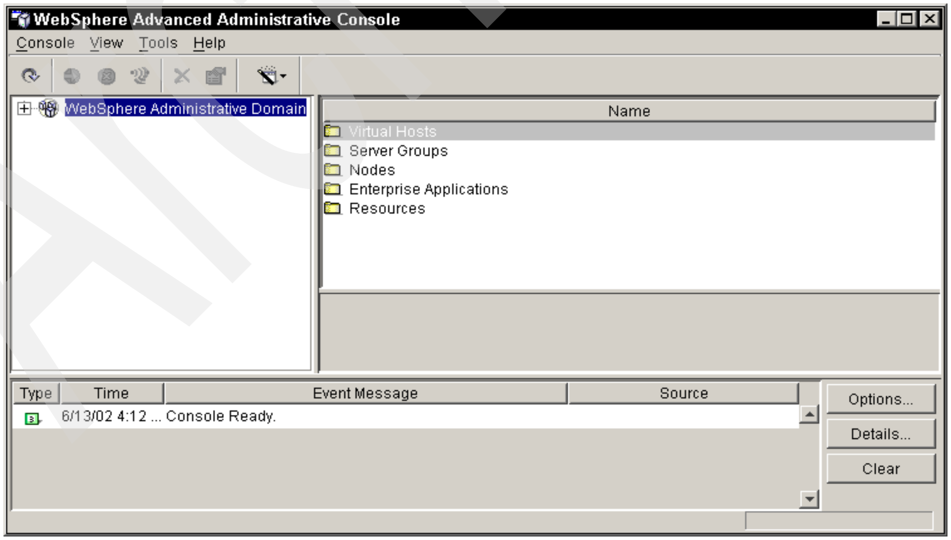


Figure 11-4 WebSphere administrative console

Default resources

We expanded the contents of the WebSphere Administrative Domain by selecting the square containing the plus sign. Expanding down through the topology we saw that all the default resources supplied with WebSphere Application Server were available. This included the default application server instance Default Server and Web modules such as default_app and examples.

11.2.2 Installing the CICS ECI resource adapter

The CICS TG provides two CICS resource adapters: the ECI resource adapter and the EPI resource adapter. The ECI resource adapter is used by CTGTesterCCI. This section describes how to install one or both of these resource adapters into WebSphere Application Server.

This section assumes that the CICS Transaction Gateway V5.0 is already installed. Please follow these steps to install the CICS resource adapters:

1. Open the WebSphere Advanced administrative console. Expand **WebSphere Administrative Domain** -> **Resources**. Notice the J2C Resource Adapters folder (Figure 11-5). This is where the resource adapters are defined to the application server.

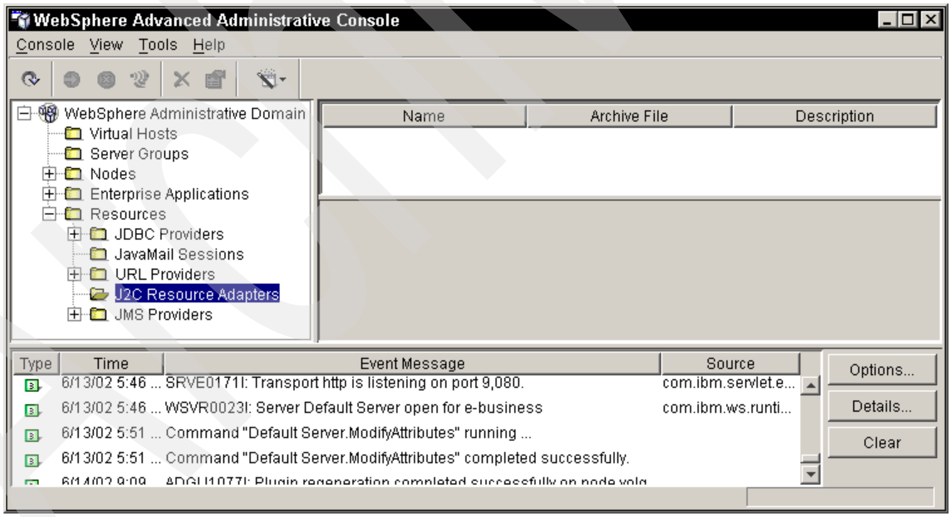


Figure 11-5 Administrative console J2C resource adapters

2. Resource adapters are installed into application servers by adding the contents of a Resource Adapter Module (represented by a file with an extension of rar) to the application server. This RAR file contains a collection of JAR files relating to the resource adapter, and a deployment descriptor

(ra.xml) that describes the deployment properties for the resource adapter. The CICS TG provides a RAR file for each CICS resource adapter. To install the CICS ECI resource adapter, follow these steps:

a. Right-click **J2C Resource Adapters** and select **New**. This opens the J2C Resource Adapter Properties window.

b. In the General tab, enter a name for the resource adapter in the Name field. Use CICS ECI.

c. In the General tab, expand the **Archive file name** textbox.

d. This opens the Install Driver window. Select the node you wish to install the resource adapter to by clicking the node name, then click the **Browse** button to select the RAR file to use.

e. To install the ECI resource adapter, move to the subdirectory deployable under the CICS TG install directory and select **cicseci.rar**. Select **Open** to select this file, then select **Install**.

f. The completed J2C Resource Adapter Properties window is shown in Figure 11-6. Select the **Connections** and **Advanced** tabs to view information about the resource adapter.

g. Select **OK** to finish. If the resource adapter was installed correctly, a window will display saying `Command "J2CResourceAdapter.create" completed successfully`.

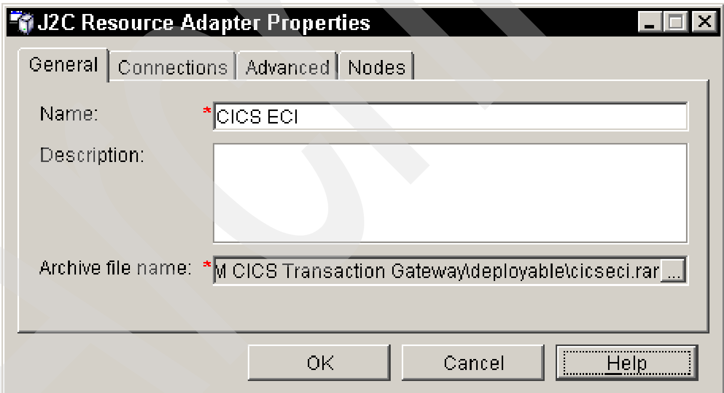


Figure 11-6 J2C resource adapter properties

3. If you wish to install the EPI resource adapter, repeat the above process, selecting **cicsepi.rar** as the archive file name.

Configure a connection factory

Every resource adapter installed in WebSphere Application Server should have one or more connection factories associated with it. A connection factory contains deployment specific connection information. In the case of the CICS resource adapters, this information includes:

- The connection URL of the CICS Transaction Gateway
- The CICS server to communicate with
- The CICS program to call, or the CICS transaction to start
- A user ID and password to flow to CICS

At least one connection factory is required to use the resource adapter in a managed environment. Setting up multiple connection factories allows you to configure a collection of possible connection options. When you deploy an enterprise bean into WebSphere Application Server, you must select which connection factory you wish to use.

This section explains how to create and configure a connection factory for the CICS ECI resource adapter. This connection factory will be used in the next section to test the J2EE Connector Architecture support in the application server.

- In order to open the Connection Factories window in the WebSphere administrative console, take the following steps:

  a. Expand **WebSphere Administrative Domain -> Resources -> J2C Resource Adapters**. This will display the resource adapter you have installed.

  b. Expand the CICS ECI resource adapter, and select **J2C Connection Factories**. A list of all connection factories created for this resource adapter (if any) will be displayed in the right pane.

- To create a new connection factory, take the following steps:

  a. Right-click **J2C Connection Factories** and select **New**. This will open the J2C Connection Factory Properties window.

  b. In the General tab enter the following:

     - In the Name field, enter a name for the connection factory. Use `SCSCPJA1 - local`.
     - Optionally, enter a path in the JNDI binding path field. We used `eis/SCSCPJA1`. If you leave this blank, a JNDI binding path will be created for you in the format of `eis/`<*connection factory name*>.
     - Notice the J2C Resource Adapter field is set to `CICS ECI`.

  – The Advanced tab allows you to set connection pooling information.

– The Connections tab allows you to set connection-specific information. Set the following (Figure 11-7):

 • ConnectionURL sets the connection URL of the CICS Transaction Gateway to use. Use `local:` if you wish to use a local CICS TG. However, if you wish to connect to a remote Gateway daemon, specify the CICS TG URL here.

 • ServerName sets the name of the CICS server to connect to. Use `SCSCPJA1`.

 • If your CICS region requires security credentials, specify values for the UserName and Password parameters.

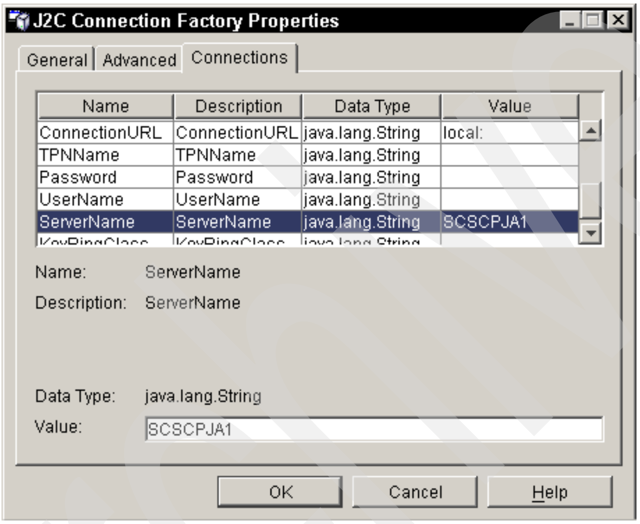


Figure 11-7 J2C connection factory properties, connections definition

– Click **OK** when finished. If the connection factory was installed correctly, a window will display saying `Command "J2CConnectionFactory.create" completed successfully`. This will publish a reference to the connection factory in the JNDI name space.

► The newly created connection factory will be displayed in the right pane (Figure 11-8 on page 299).

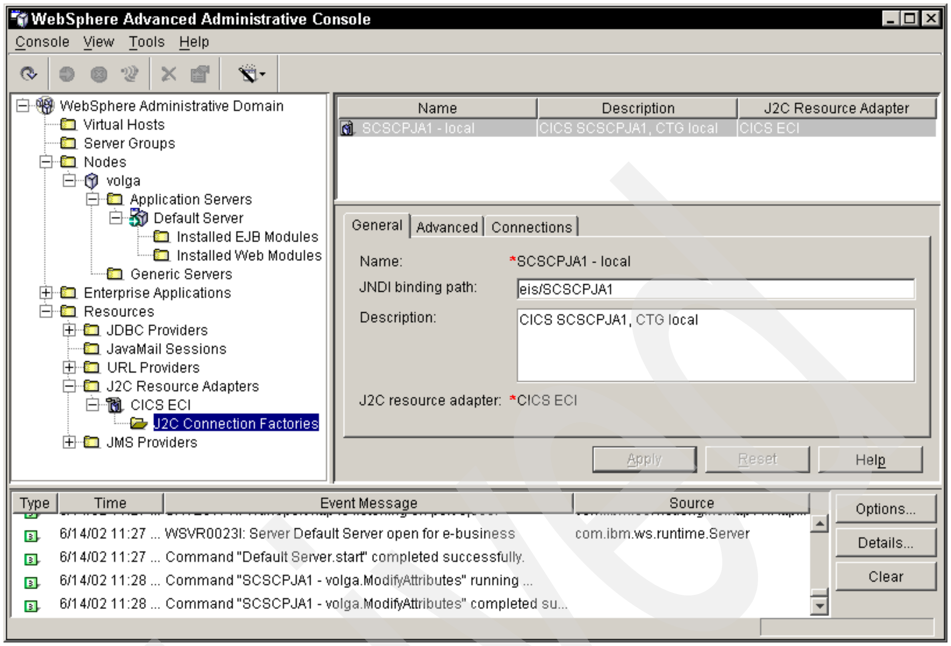


Figure 11-8 Administrative console with the Connection Factory

> **Note:** To remove a connection factory definition, all enterprise applications that use it must first be stopped and removed. Also to remove the ECI resource adapter, all connection factory definitions must first be removed.

11.2.3 Deploying our sample enterprise applications

We deployed our sample enterprise applications, CTGTesterECI and CTGTesterCCI. You can obtain these samples with the additional material supplied with this book. For further details, refer to Appendix C, “Additional material” on page 389.

Deploying the CTGTesterECI application

To deploy our CTGTesterECI servlet we first copied the J2EE enterprise archive (EAR) file CTGTesterECI.ear onto our Windows 2000 machine into the directory c:\WebSphere\AppServer\installableApps. We then had to install the enterprise application in WebSphere using the administrative console.

To install our enterprise application using the administrative console, perform the following steps:

1. Select **Console -> Wizards -> Install Enterprise Application**.
2. This opens the Install Enterprise Application Wizard window. Select **Install Application (*.ear)** then select **Browse**. Move to the directory where you put CTGTesterECI.ear, select this file, then select **Open**. The window will look like Figure 11-9.

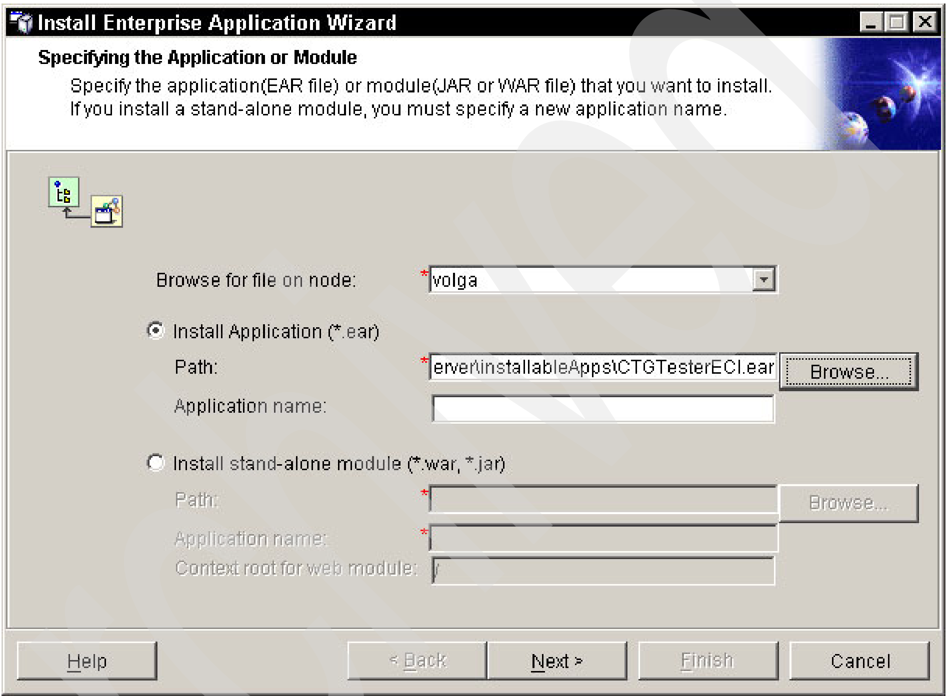


Figure 11-9 Installing CTGTesterECI into WebSphere Application Server

3. You should use the default values specified in the EAR file to deploy this enterprise application. Click **Next** continually until the Selecting Virtual Hosts for Web Modules window appears (Figure 11-10 on page 301).

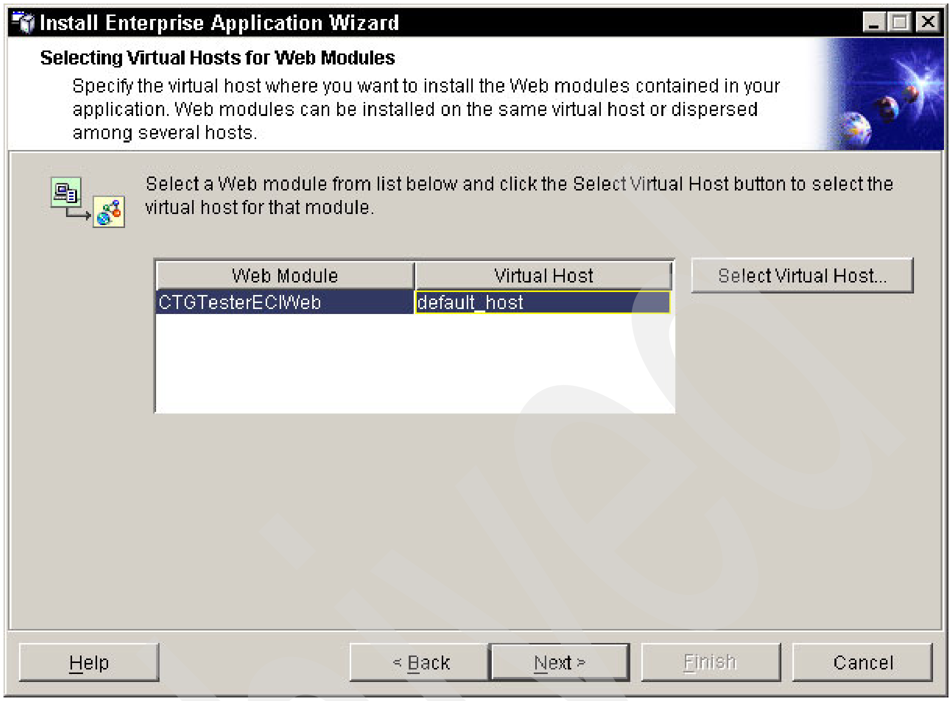


Figure 11-10 Selecting Virtual Hosts for Web Modules window for CTGTesterECI

4. On the Selecting Virtual Hosts for Web Modules window, ensure that the Virtual Host is set to default_host, then click **Next**.
5. Click **Next** past the Selecting Application Servers window.
6. Click **Finish** at the Completing the Application Installation window. If the enterprise application was installed successfully, a window will display saying `Command "EnterpriseApp.install" completed successfully`.

The enterprise application CTGTesterECI will be created. To view it in the administrative console, expand **WebSphere Administrative Domain -> Enterprise Applications**. See Figure 11-11 on page 302.

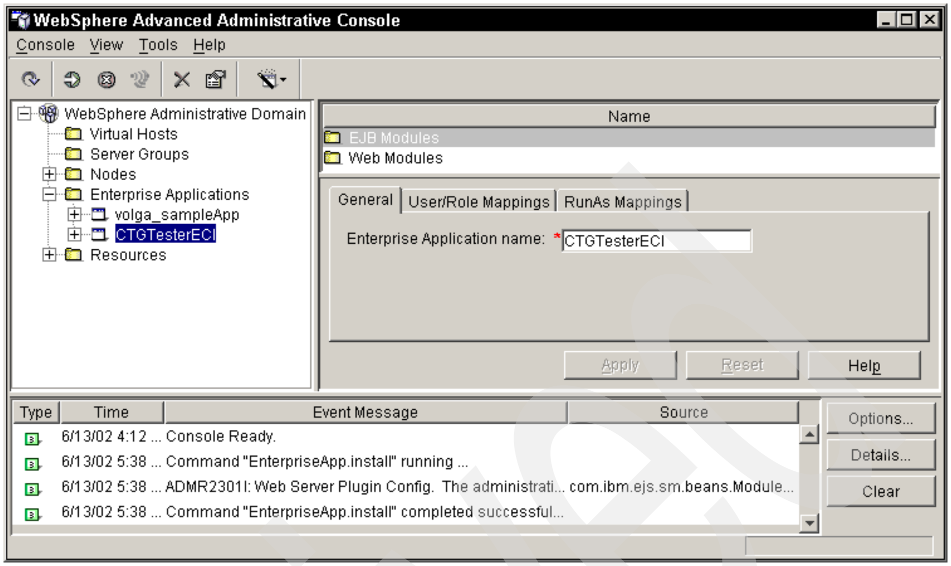


Figure 11-11 Viewing the installed enterprise application CTGTesterECI

In order to use the CTGTesterECI application, the CICS TG Java class library (ctgclient.jar) needs to be added to the application server's CLASSPATH. Additionally, ctgserver.jar is needed for local mode, since the CICS TG JavaGateway class must be instantiated within the Web application, as there is no Gateway daemon in use.

1. From the WebSphere administrative console, expand **WebSphere Administrative Domain -> Nodes ->** ***node name*** **-> Application Servers**. Click **Default Server** (Figure 11-12 on page 303).

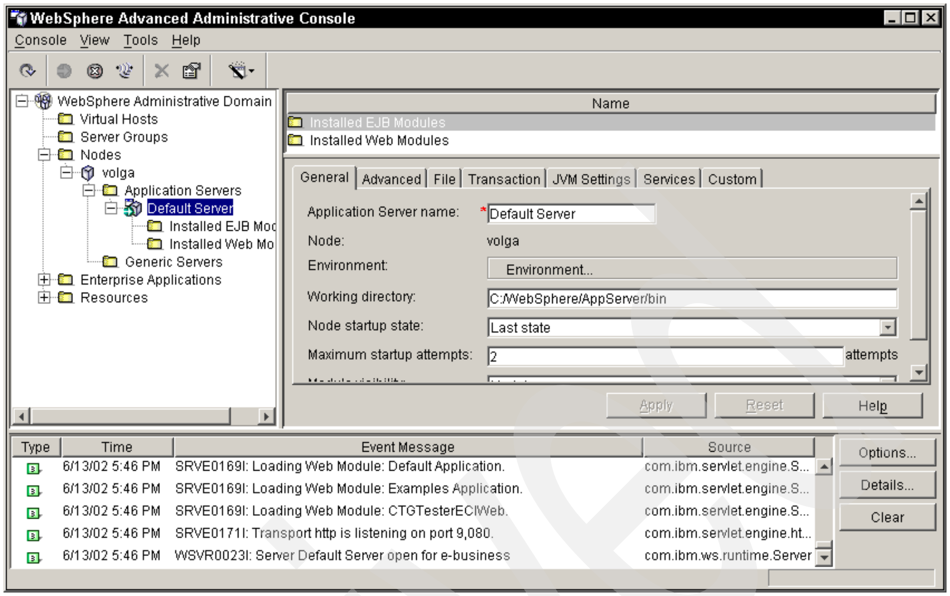


Figure 11-12 Default server configuration

2. Select the **JVM Settings** tab. On the Classpaths section of the tab, click **Add** and enter the absolute path to ctgclient.jar. Click **Add** again and enter the absolute path to ctgserver.jar. These JARs are located in the CICS TG classes directory.
3. Click **Apply**. The settings will be applied to the server configuration; see Figure 11-13.

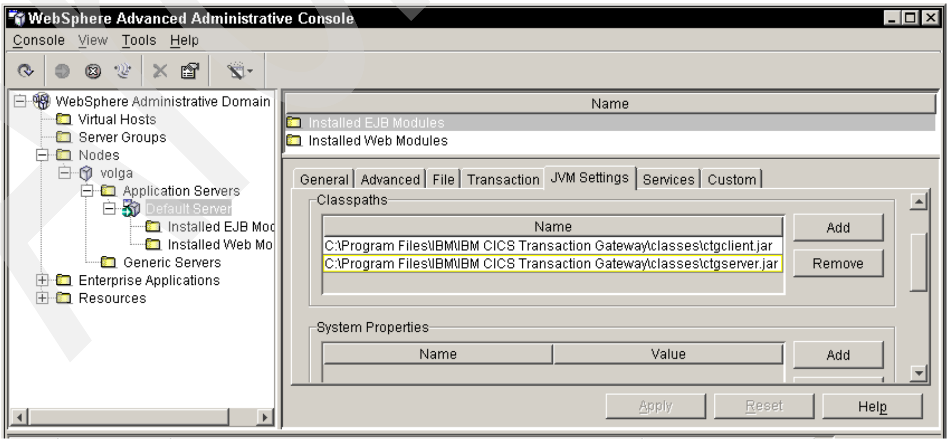


Figure 11-13 Adding CICS TG Java classes to the default server CLASSPATH

4. The Web server plugin must now be regenerated. From the administrative console, expand **WebSphere Administrative Domain -> Nodes** and right-click your node name. Select **Regen Webserver plugin**.
5. Before running this enterprise application, restart the default application server. To do this, expand **WebSphere Administrative Domain -> Nodes ->** ***node name*** **-> Application Servers**. Right-click **Default Server**. Stop the server if it is running, then right-click **Default Server** and select **Start** to restart the default server.

Deploying the CTGTesterCCI application

To deploy our CTGTesterCCI application, we first copied the J2EE enterprise archive (EAR) file CTGTesterCCI.ear onto our Windows 2000 machine into the directory c:\WebSphere\AppServer\installableApps. We then had to install the Enterprise Application in WebSphere using the administrative console.

To install our enterprise application using the administrative console, perform the following steps:

1. Select **Console -> Wizards -> Install Enterprise Application**.
2. This opens the Install Enterprise Application Wizard window (Figure 11-14). Select **Install Application (*.ear)** then select **Browse**. Navigate to the directory containing CTGTesterCCI.ear, select this file, then select **Open**.

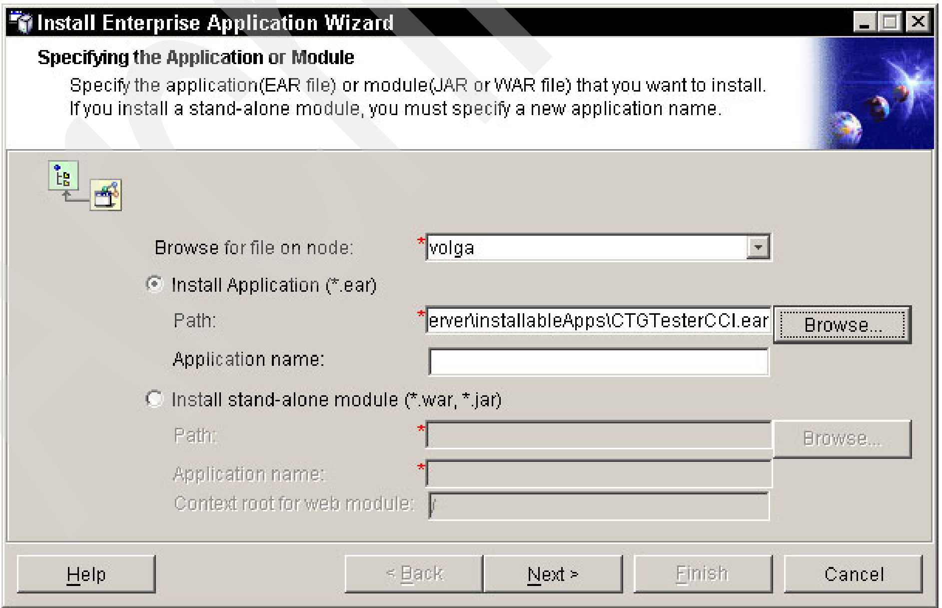


Figure 11-14 Installing CTGTesterCCI into WebSphere Application Server

3. Click **Next** three times to display the Binding Enterprise Beans to JNDI Names window (Figure 11-15). This allows you to change the JNDI name under which the session bean is bound in the JNDI name space. The JNDI name will already be set to `ejbs/CTGTesterCCI`, as specified in the EAR file. Click **Next** to accept this value.

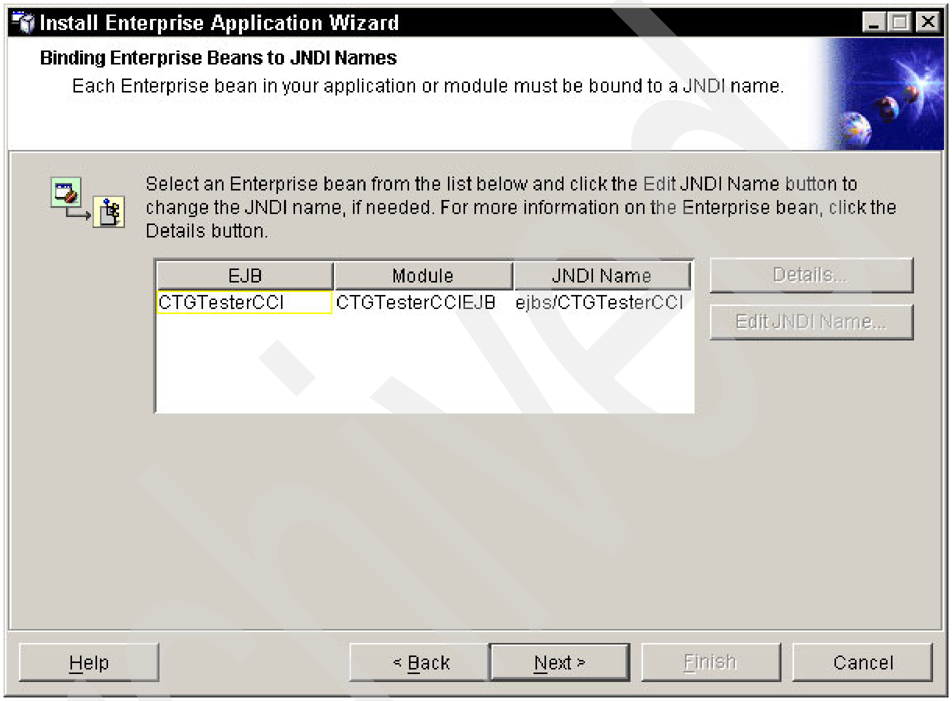


Figure 11-15 Binding Enterprise Beans to JNDI Names window

4. The Mapping EJB References to Enterprise Beans window is shown (Figure 11-16 on page 306). This allows you to change the JNDI name the servlet's EJB reference is bound to. The JNDI name will already be set to ejbs/CTGTesterCCI, as specified in the EAR file. This matches the JNDI name of the session bean, seen in the previous window. Click **Next** to accept this value.

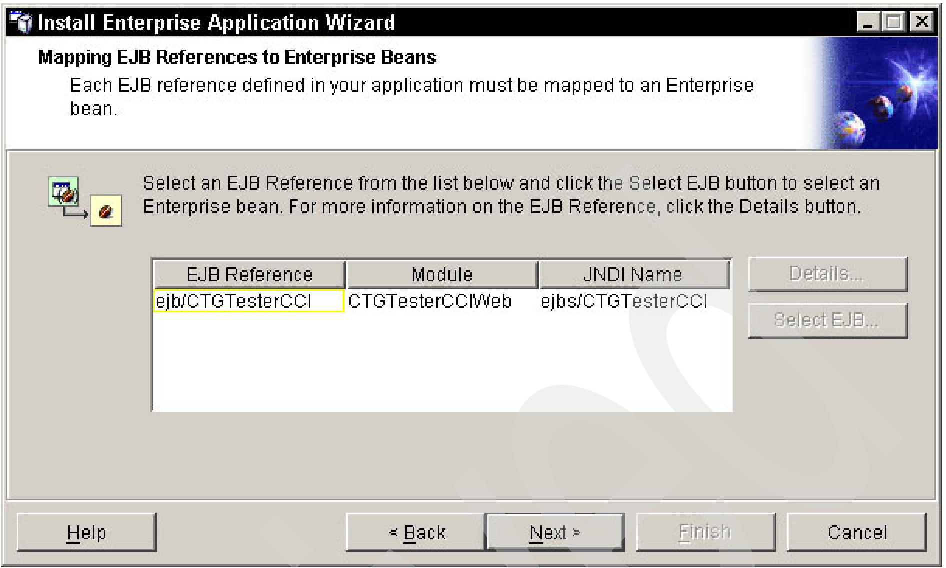


Figure 11-16   Mapping EJB References to Enterprise Beans window

5. The Mapping Resource References to Resources window is shown. Click **Next**. This will result in the error message shown in Figure 11-17.

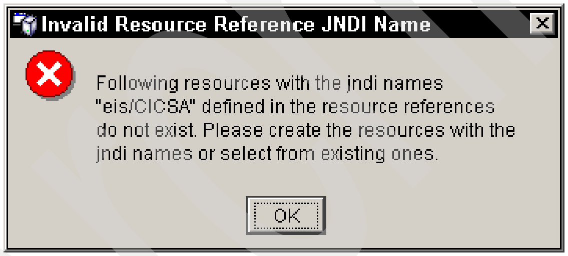


Figure 11-17   Invalid Resource Reference

This error arises because the JNDI binding name eis/CICSA that is specified for the ECI resource reference in the EAR does not match the JNDI name of an existing connection factory resource in WebSphere, since at this point we specify a JNDI name of a resource that does exist. We suggest that you use the connection factory that was configured previously with the JNDI name eis/SCSCPJA1. To do this, click **OK** and follow the remaining instructions.

6. Click the **Select Resource** button, and choose the name of the connection factory resource that you want to use. The name of the resource in WebSphere, rather than its JNDI name, is displayed. Choose the **SCSCPJA1 - local** resource that was previously created. Click **OK** to show the completed window (Figure 11-18) and click **Next**.

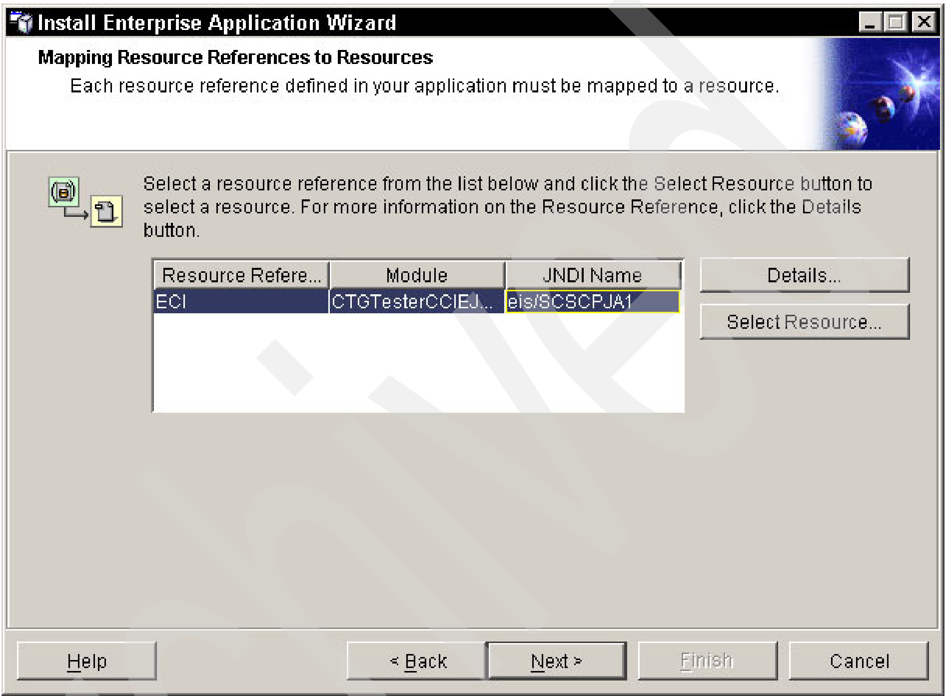


Figure 11-18 Using previously created connection factory

7. Click **Next** twice. On the Selecting Virtual Hosts for Web Modules window ensure that the Virtual Host is set to default_host, then click **Next**.
8. Click **Next** past the Selecting Application Servers window.
9. Click **Finish** at the Completing the Application Installation window. When prompted whether to regenerate the application code for installation, choose **No**. If the enterprise application was installed successfully, a window will display saying `Command "EnterpriseApp.install" completed successfully.`

The enterprise application CTGTesterCCI is now created. To view it in the administrative console, expand **WebSphere Administrative Domain -> Enterprise Applications**. See Figure 11-19.

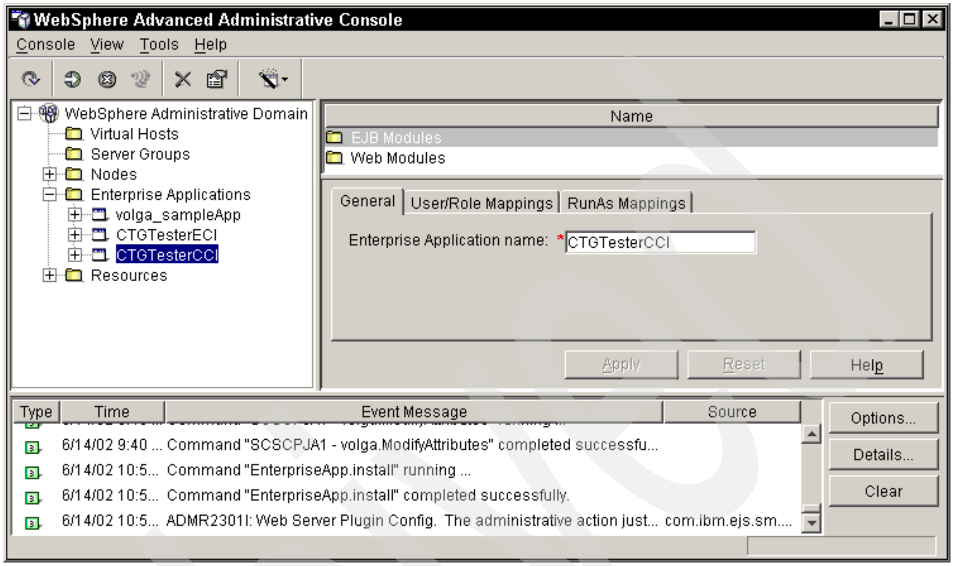


Figure 11-19 Viewing the installed enterprise application CTGTesterCCI

10. The Web server plugin must now be regenerated. From the administrative console, expand **WebSphere Administrative Domain -> Nodes** and right-click your node name. Select **Regen Webserver plugin**.

11. Before running this enterprise application, restart the default application server. To do this, expand **WebSphere Administrative Domain -> Nodes ->** ***node name*** **-> Application Servers**. Right-click **Default Server**. Stop the server if it is running, then select **Start** to restart the default server.

11.3 Testing the configuration

After we installed and configured the software components we tested our WebSphere for Windows configuration in two scenarios using a local CICS TG, and one scenario using a remote CICS TG.

11.3.1 Local testing

We tested two scenarios using the local: protocol. The local: protocol bypasses the Gateway daemon and instead causes the application to invoke the native Client daemon libraries using the Java Native Interface (JNI) directly.

Result with ECIRequest/servlet application

First we used our ECIRequest-based servlet application, CTGTesterECI, to call the CICS program EC01, using a local CICS TG (Figure 11-20).

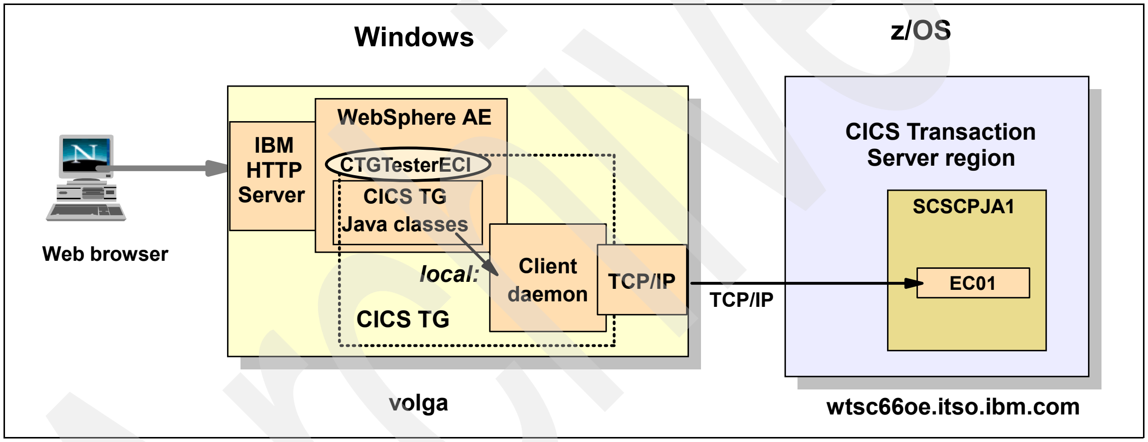


Figure 11-20 WebSphere with CTGTesterECI using local CICS TG

To verify this configuration, we performed the following tasks:

1. We started the IBM HTTP Server by clicking **Start -> Programs -> IBM HTTP Server -> Start HTTP Server**.
2. We opened our Web browser and loaded the welcome page of the IBM HTTP Server using the URL `http://volga`. This allowed us to verify that the HTTP server was correctly serving Web pages.
3. We started the Client daemon TCP/IP connection to the CICS Transaction Server region using the Windows command `CICSCLI -S=SCSCPJA1`.

   To verify the status of the Client daemon connection, we used the command `CICSCLI -L`, the output of which is shown in Example 11-1 on page 310.

Example 11-1 Viewing the status of the Client daemon connection

```
C:\>CICSCLI -L
CCL8001I CICSCLI - CICS Client Control Program
CCL0002I (C) Copyright IBM Corporation 1994,2001.  All rights reserved.
CCL8041I The CICS Client is using the following servers:
CCL8042I Server 'SCSCPJA1' (using 'TCPIP' to 'wtsc66oe.itso.ibm.com') is
available
```

For more details on configuring the TCP/IP protocol with the CICS TG, refer to Chapter 5, “TCP/IP connections to CICS” on page 81.

4. To start our WebSphere Application Server, we selected **Start -> Programs -> IBM WebSphere -> Application Server V4.0 AE -> Start Admin Server**. We also started the administrative console to allow us to view the status of the resources in our WebSphere domain. We started this using the path **Start -> Programs -> IBM WebSphere -> Application Server V4.0 AE -> Administrator’s Console**.

   The default server in our WebSphere administrative domain was started, as indicated by the green disk beside the server instance.

5. To determine if the CTGTesterECI application was running, from the administrative console we expanded **WebSphere Administrative Domain -> Enterprise Applications** and right-clicked the **CTGTesterECI** node. We selected **Show Status** from the pop-up menu. The resulting pop-up window showed that the CTGTesterECI Web application was running on the default server (Figure 11-21).

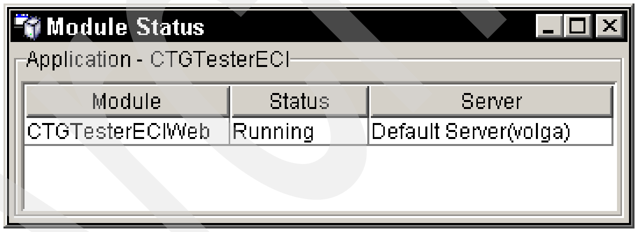


Figure 11-21 Status window showing the CTGTesterECI application running

> **Note:** Each individual application can be started by right-clicking their node and selecting **Start** from the pop-up menu. The default server can also be restarted by stopping and starting it. Starting a server will also start all applications that run inside it.

6. We opened our Web browser and loaded the welcome page of the CTGTesterECI application using the URL `http://volga/CTGTesterECIWeb/`. This displayed the page shown in Figure 11-22 on page 311. The default

parameters were set up to cause the CTGTesterECI servlet to make a ECI call to EC01 on our CICS region SCSCPJA1, using the local: protocol.

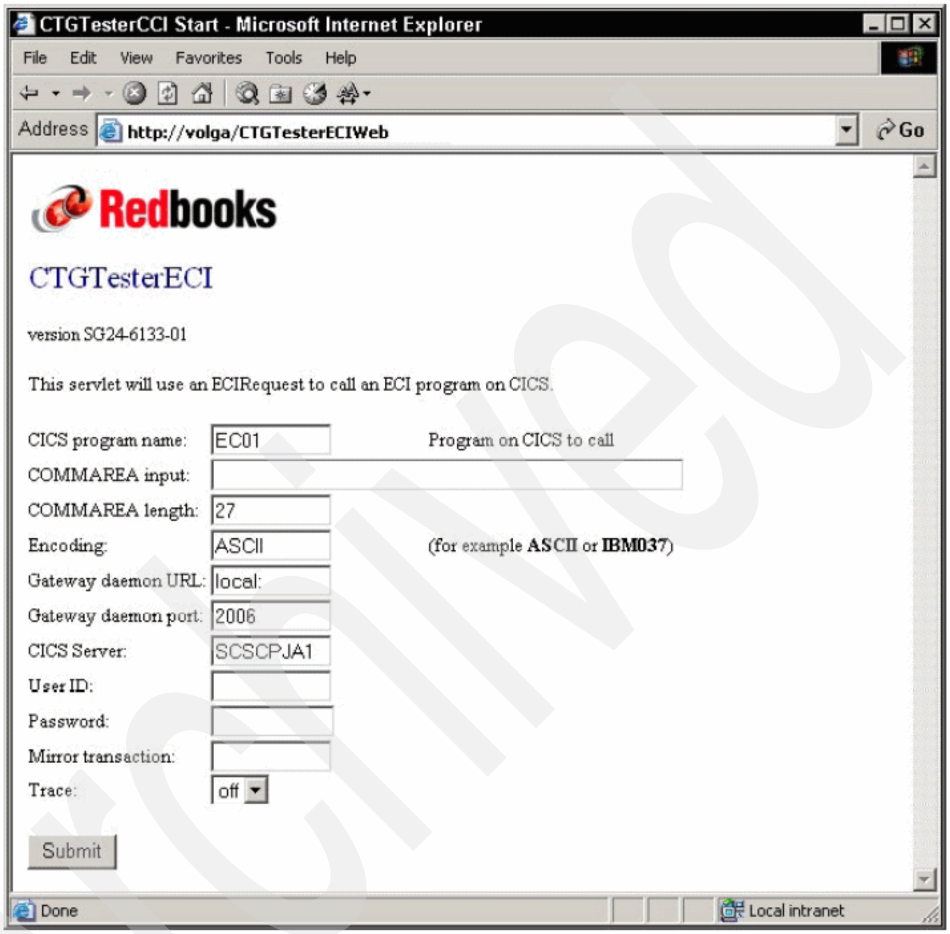


Figure 11-22 CTGTesterECI enterprise application welcome page

7. We clicked the **Submit** button. This used the ECIRequest Java classes to call the EC01 CICS program. The successful response was shown in the Web browser. See Figure 11-23 on page 312.

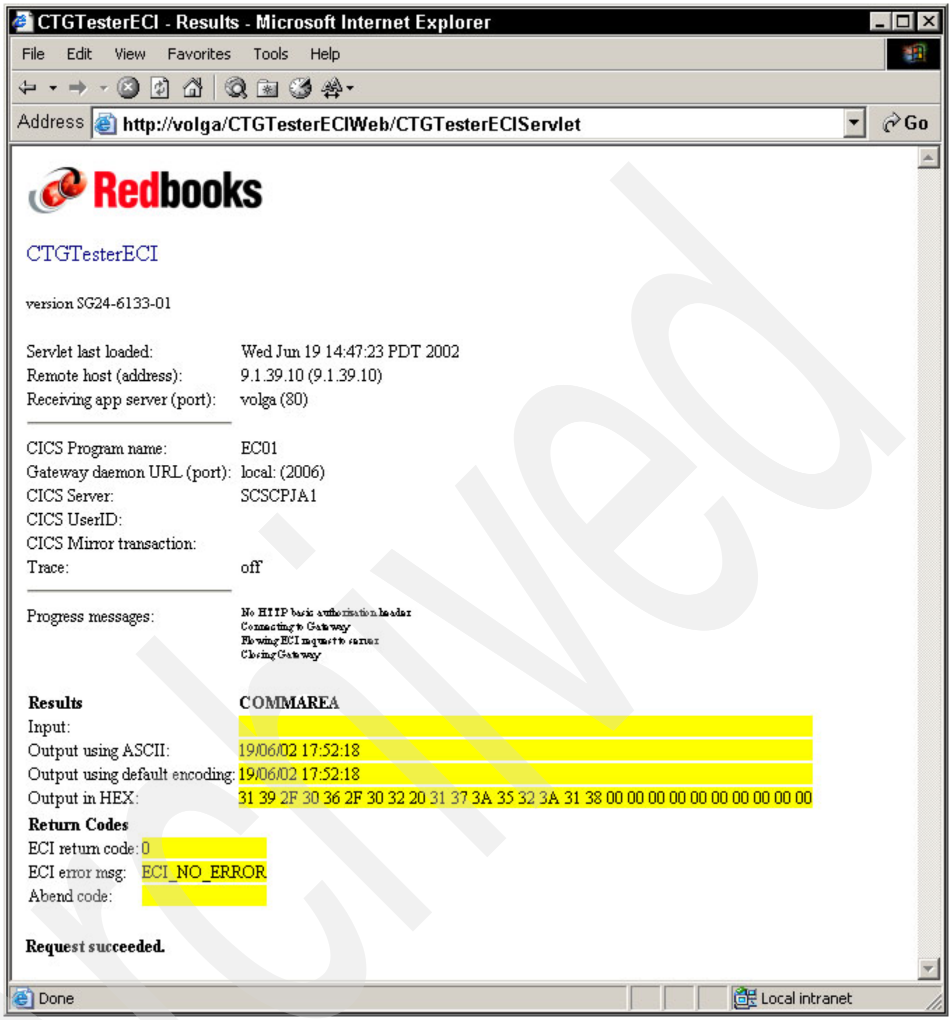


Figure 11-23 CTGTesterECI enterprise application response

For more details on how the CTGTesterECI application functions, refer to Appendix B, “Sample applications” on page 337. For instructions on how to obtain the sample code for CTGTesterECI, refer to Appendix C, “Additional material” on page 389.

Results with J2EE application

Next we used our J2EE application, CTGTesterCCI, to call the CICS program EC01, using the CICS TG ECI resource adapter and the connection factory we set up previously. This used a local CICS TG (Figure 11-24).

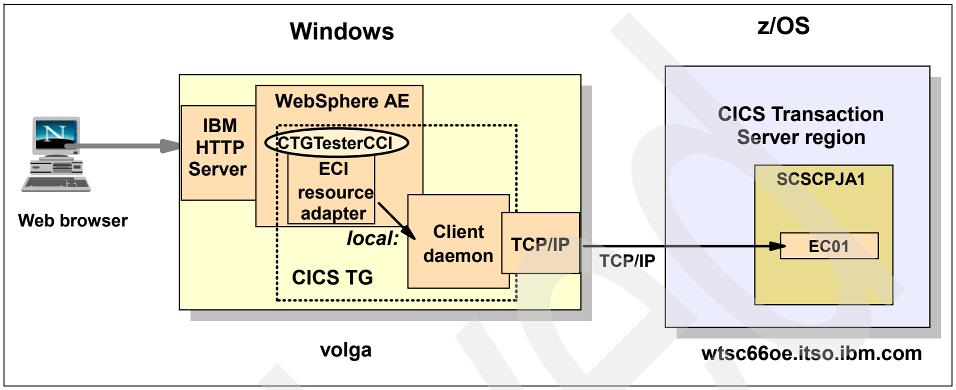


Figure 11-24 WebSphere with CTGTesterCCI using local CICS TG

To verify this configuration we performed the following steps:

1. We verified that the IBM HTTP Server was running using the steps detailed in the previous section.
2. We verified that the WebSphere default server was running and that our application CTGTesterCCI was running using the steps detailed in the previous section.
3. We opened our Web browser and loaded the welcome page of the CTGTesterCCI application using the URL `http://volga/CTGTesterCCIWeb/`. This displayed the page shown in Figure 11-25 on page 314. The default parameters were set up to cause the CTGTesterCCI application to make an ECI call to EC01 on a CICS region, using the ECI resource adapter. The ECI resource adapter was configured to use the local: protocol and the server SCSCPJA1, defined in “Configure a connection factory” on page 297.

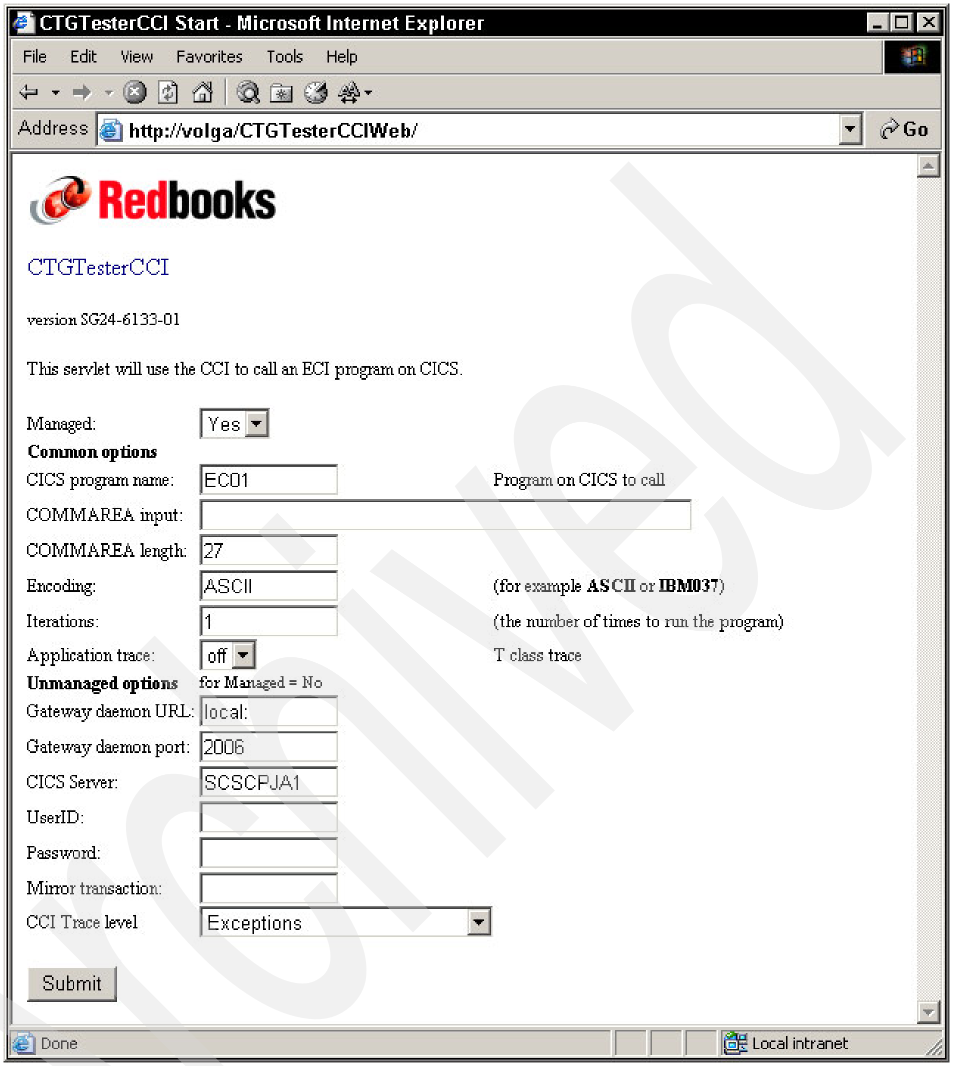


Figure 11-25 CTGTesterCCI enterprise application welcome page

4. We clicked the **Submit** button. This used the CICS TG resource adapter to call the EC01 CICS program. The successful response was shown in the Web browser. See Figure 11-26 on page 315.

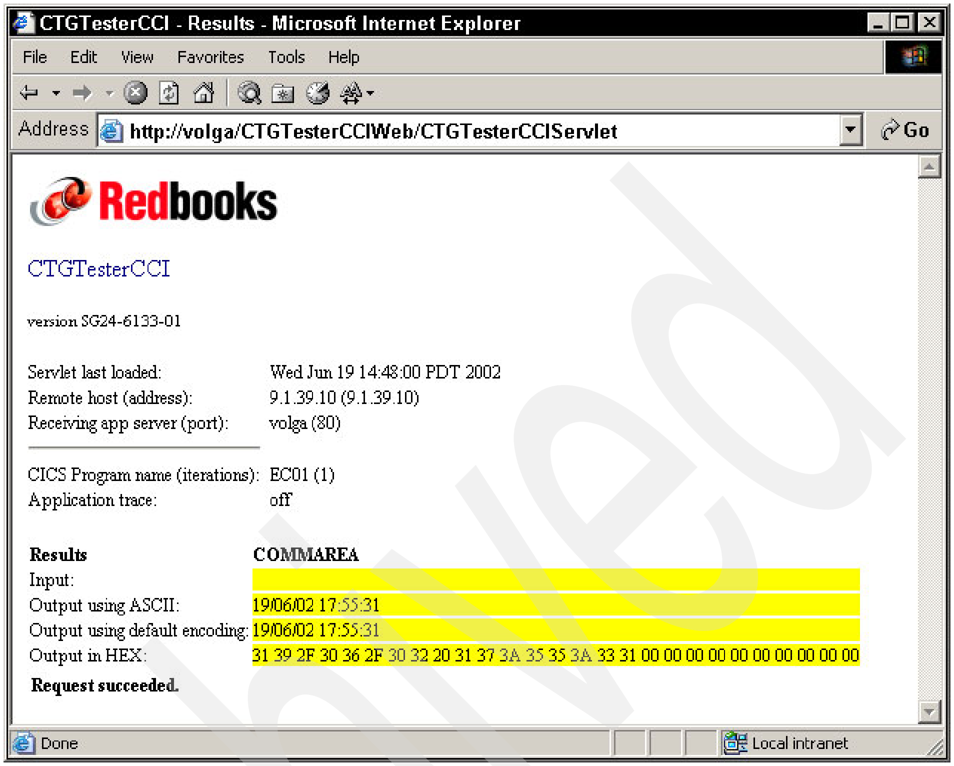


Figure 11-26 CTGTesterCCI enterprise application response

For more details on how the CTGTesterCCI application functions, refer to Appendix B, “Sample applications” on page 337.

11.3.2 Remote testing

We next tested a remote scenario using the tcp: protocol to connect to a remote Gateway daemon running on z/OS. The tcp: protocol is used to make a socket connection from the Windows machine to the Gateway daemon, which then forwards the request onto a connected CICS region using the EXCI.

Results with J2EE application

We used our J2EE application, CTGTesterCCI, to call the CICS program EC01, using the CICS TG SCSCG5 on our z/OS system (Figure 11-27).

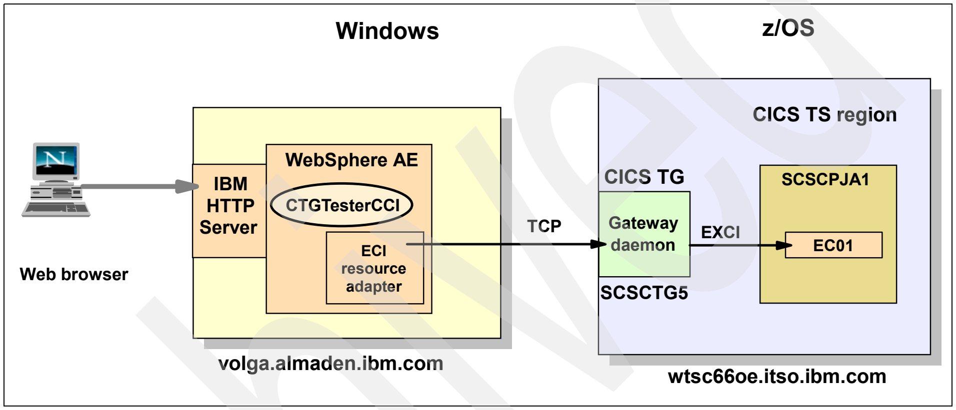


Figure 11-27 WebSphere with CTGTesterCCI using remote CICS TG

To configure CTGTesterCCI to use a remote CICS TG, we configured a new connection factory, following the steps in “Configure a connection factory” on page 297. We specified the following settings:

Name	SCSCPJA1 - scsctg5
JNDI binding path	SCSCPJA1-scsctg5
ConnectionURL	tcp://wtsc66oe.itso.ibm.com
ServerName	SCSCPJA1
PortNumber	2006

We then configured the CTGTesterCCI enterprise application to use this connection factory instead of SCSCPJA1 - local. We performed the following steps:

1. In the administrative console, we expanded **WebSphere Administrative Domain -> Enterprise Applications -> CTGTesterCCI**. We clicked **EJB Modules** under CTGTesterCCI. We clicked the **Resource Reference** tab (Figure 11-28 on page 317).

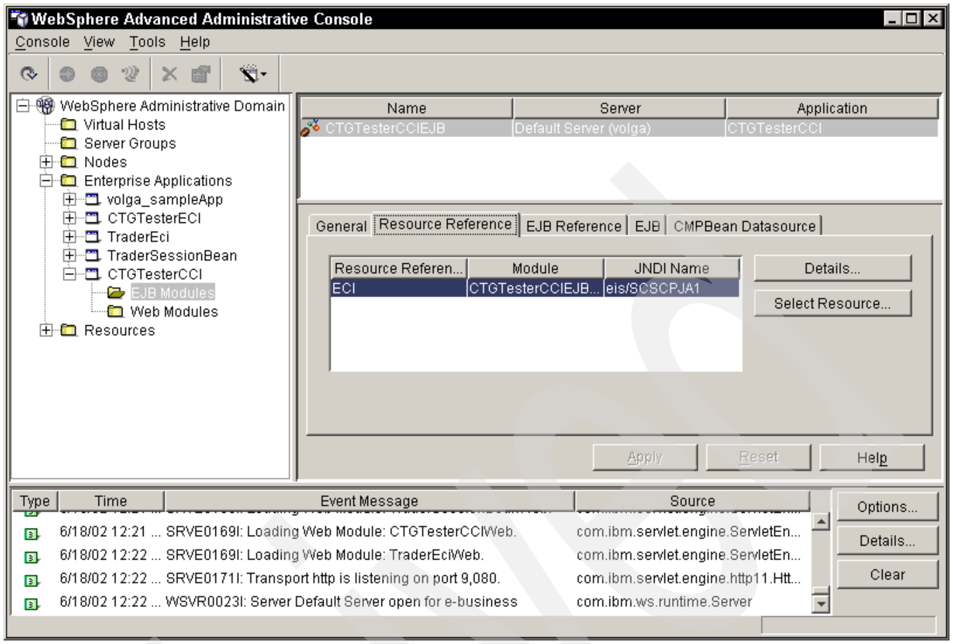


Figure 11-28 CTGTesterCCI resource references

2. We selected the **ECI** resource reference. We clicked the **Select Resource** button. This caused the Select Resource pop-up window to appear. From this window, we changed the resource in the drop-down list to `SCSCPJA1-scsctg5` (Figure 11-29). We then clicked **OK**.

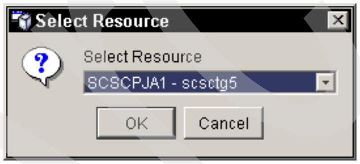


Figure 11-29 Drop-down list

3. In the Resource Reference window, we clicked the **Apply** button.
4. The CTGTesterCCI enterprise application had to be restarted to use the new connection factory. We right-clicked the **CTGTesterCCI** node under Enterprise Applications and selected **Stop**. Once CTGTesterCCI had stopped, we right-clicked the **CTGTesterCCI** node again and selected **Start**.

We then retested CTGTesterCCI following the steps in “Results with J2EE application” on page 316. The results were similar to those shown in

Figure 11-26 on page 315, because the resource adapter hides details of the CICS server and CICS TG protocol it uses.

Important: We found that if no transactional attribute is specified on a session bean deployed into WebSphere AE, then the WebSphere still makes the bean transactional by default. If this bean makes a call to the CICS TG resource adapter, this will be run under an extended LUW. If used with a remote CICS TG on z/OS, the CICS TG must be configured with transactional EXCI support enabled; otherwise a simple request would fail with the error CTG9631E. For further details about enabling transactional EXCI, refer to page 152 in Chapter 7.

11.4 Problem determination

In this section, we document useful information we learned while configuring this scenario and further information on problem determination and tracing.

11.4.1 Tips and utilities

In this section, you will find commands and utilities for debugging problems with your configuration. You will also find some additional information about topics discussed in this chapter.

Serious events

The Event Viewer is shown in the bottom part of the WebSphere administrative console (see Figure 11-30). This Viewer displays records of the last several serious events to occur since the administrative server was started, as well as warning and audit messages.

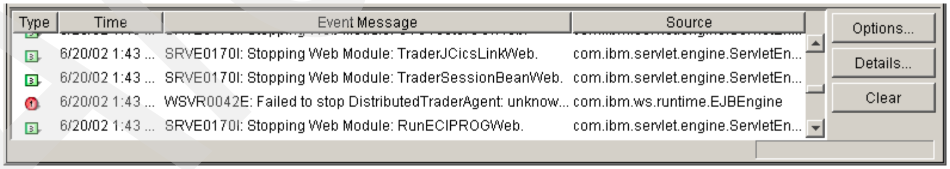


Figure 11-30 Event Viewer window on the administrative console

Use the Options window to specify Event Viewer properties, such as how many event records to keep at one time and how often to update the messages on the administrative console with data from the administrative server. The Log Limit property specifies how many serious event records to keep. They are stored in the administrative database. If the administrative database is getting too full, set the Log Limit to the minimum value that is reasonable for your environment.

WebSphere logs

Log files provide information about administrative and application servers as they initialize and run. After an error or problem condition occurs, logs can be reviewed for clues as to what happened. The logs provided by WebSphere include stderr, stdout, wasdb2, and wssetup logs and are located in the \WebSphere\AppServer\logs directory.

- Stderr and stdout logs

  These capture events presented through two of the three standard I/O streams. Stdin are the arguments entered with a command or program. Stdout is the output displayed to the user. Stderr are the errors thrown by the code.

  The stdout and stderr logs contain application server communication, for application servers and the admin server, providing error and informational messages that reflect server startup and server status change requests, such as start, stop, and restart. Entries displayed in the Event Viewer are written to the standard out log.

  Output from `System.err.println` and `System.out.println` statements in the application code will also appear in the application server stdout and stderr logs respectively.

  By default, logs for the default server are output to the files Default_Server_stdout.log and Default_Server_stderr.log. The file name and destination directory of the logs can be specified in the File tab of the default server node (Figure 11-31).

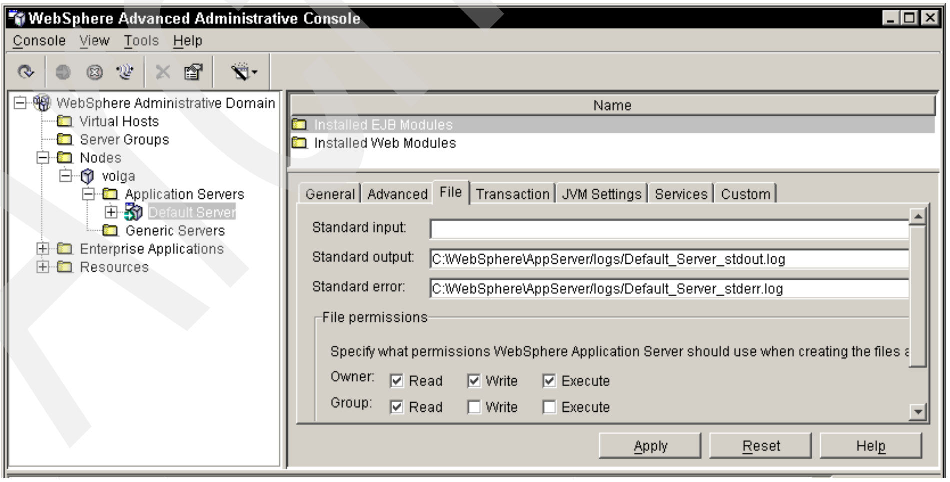


Figure 11-31 Specifying file destinations for server logs

- Wasdb2 log

  This log is created when the DB2 database is configured. This log enables you to determine if the WebSphere database was created correctly and if WebSphere Application Server can connect to it. Errors creating the system management repository tables will be logged in this file, called wasdb2.log.

- Wssetup log

  This log is created during the install process and shows if the install process was successful. It includes installation processes such as verifying prerequisite, downloading files, and updating configuration files. The file is called wssetup.log.

Common errors

If you see the following exception message when executing a request against a remote CICS TG for z/OS using the resource adapter:

```
javax.resource.spi.ResourceAdapterInternalException: CTG9631E: Error
occurred during interaction with CICS. Error Code=ECI_ERR_SYSTEM_ERROR
```

and the following message in the CICS TG JNI trace:

```
[020e06ed,10db6250,Worker-0    ] : CCL6818E: CcicsECI: Begin Context
failed. RRM Return code = 1878.
```

Check that transactional EXCI is enabled on the CICS TG for z/OS. See 11.3.2, “Remote testing” on page 316 for further details.

Altering connection factory settings

We found that when changing settings on existing connection factory definitions, for example adding a user name and password, we had to restart the default server for the settings to take effect. Restarting the enterprise application that was using the connection factory was not sufficient.

Useful commands

We found the WebSphere utility dumpnamespace useful to verify that our session bean and connection factories had been registered in the JNDI name space. The utility is supplied with WebSphere in the AppServer\bin directory.

We ran the dumpnamespace utility from the WebSphere Appserver\bin directory on our Windows 2000 workstation. The utility connected to the JNDI server on the machine and output the contents of the JNDI name space, as shown in Example 11-2 on page 321.

Example 11-2 Output from dumpnamespace

```
==============================================================================
Beginning of Name Space Dump
==============================================================================

1 (top)
...
8 (top)/eis                javax.naming.Context
9 (top)/eis/SCSCPJA1       SCSCPJA1
...
57 (top)/ejbs              javax.naming.Context
58 (top)/ejbs/CTGTesterCCI itso.cics.eci.j2ee.testercci._CTGTesterCCIHome_Stub
...
```

Recall that we defined each connection factory's JNDI binding path as being eis/<connection factory name>. Example 11-2 shows our connection factory SCSCPJA1 bound with the JNDI path eis/SCSCPJA1 as specified in "Configure a connection factory" on page 297.

Also recall that when we deployed the CTGTesterCCI enterprise application, we used the value of `ejbs/CTGTesterCCI` for the JNDI Name of our CTGTesterCCI session bean. Example 11-2 also shows the JNDI entry for our session bean.

11.4.2 Tracing

Various levels of CICS TG tracing can be enabled from WebSphere. They are:

- Java application trace
- Resource adapter trace
- JNI trace

Java client trace

CICS TG Java client tracing can be set programmatically by adding the following statements to the Java application:

```
com.ibm.ctg.client.T.setDebugOn(true);
com.ibm.ctg.client.T.setTimingOn(true);
```

Both of these traces go to the stderr output stream, so they will appear in the standard error log file.

> **Note:** Enabling CICS TG Java client tracing in an application will enable it for all applications running in the application server, as a consequence of the static nature of the T class.

J2EE resource adapter trace

Resource adapter tracing can either be set programmatically, or by using the WebSphere administrative console to change properties in the connection factory.

Programmatic trace control

Resource adapter tracing is set programmatically using the `ECIManagedConnectionFactory` object. Two steps must be taken to enable tracing:

1. Turn logging on through the `setLogWriter()` method.
2. Specify the level of tracing using the `setTraceLevel()` method as shown:

```
ECIManagedConnectionFactory mcf = new ECIManagedConnectionFactory();
mcf.setLogWriter(new java.io.PrintWriter(System.out));
mcf.setTraceLevel(new Integer(mcf.RAS_TRACE_INTERNAL));
```

Valid levels of tracing are as follows, with each level of trace building upon the previous level. Therefore, ENTRY_EXIT includes everything in ERROR_EXCEPTION, and INTERNAL includes all trace levels:

RAS_TRACE_OFF	Disable all tracing
RAS_TRACE_ERROR_EXCEPTION	Output exception trace stacks
RAS_TRACE_ENTRY_EXIT	Output method entry and exit stack traces
RAS_TRACE_INTERNAL	Output debug trace entries

Tip: We found that the CICS resource adapter trace traced the execution of methods only within the resource adapter itself. The resource adapter acts as a client to the CICS TG classes, so enabling CICS TG Java client tracing will provide extra information. We suggest that you combine J2EE resource adapter trace with the use of the `com.ibm.ctg.client.T` class, since this will provide end-to-end debug information.

Managed connections

Tracing on a connection factory is set using the TraceLevel property of the connection factory resource. To modify tracing, from the WebSphere administrative console expand **WebSphere Administrative Domain -> Resources -> J2C Resource Adapters -> CICS ECI** and click **J2C Connection Factories**. Click the connection factory instance and then click the **Connections** tab. The TraceLevel property controls the tracing (Figure 11-32 on page 323).

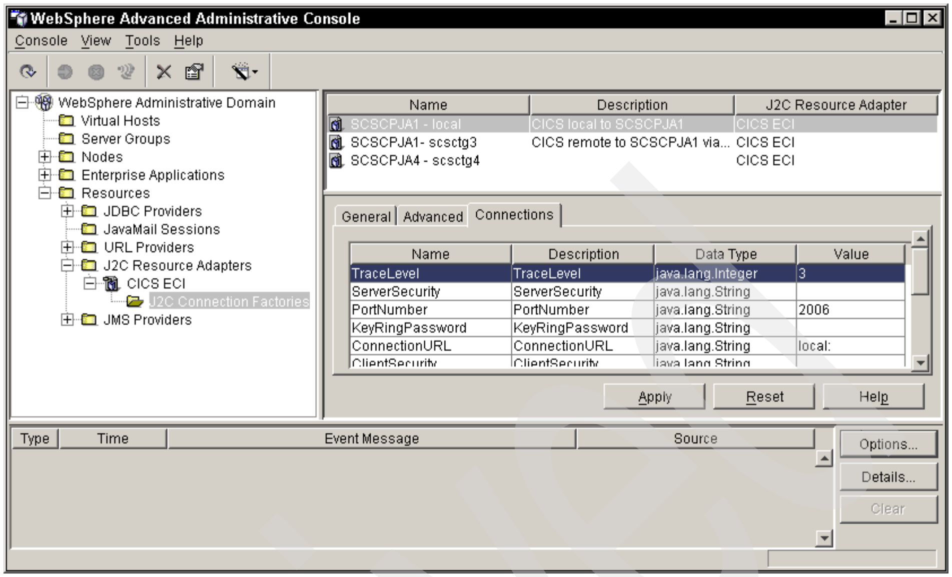


Figure 11-32 Connection factory trace setting

The same levels of tracing as for programmatic control are available. These are:

0 Disable all tracing
1 Output exception trace stacks
2 Output method entry and exit stack traces
3 Output debug trace entries

For example, to set a connection factory to trace method entry, exit and exceptions, enter `2` into the TraceLevel field.

By default the trace output goes to the destination specified in the `setLogWriter()` call in application code, and is not output if this method call is not made. In order to make the trace output go to standard error without programmatic intervention, the Java system property `com.ibm.connector2.cics.outputerr` has to be set on the application server. To do this, perform the following steps:

1. From the WebSphere administrative console, expand **WebSphere Administrative Domain -> Nodes -> *node name* -> Application Servers**. Click **Default Server.**
2. Click the **JVM Settings** tab.
3. In the System Properties section of the JVM Settings tab, click the **Add** button. In the Name field enter `com.ibm.connector2.cics.outputerr`. In the Value field, enter `true`. Click **Apply**. See Figure 11-33 on page 324.

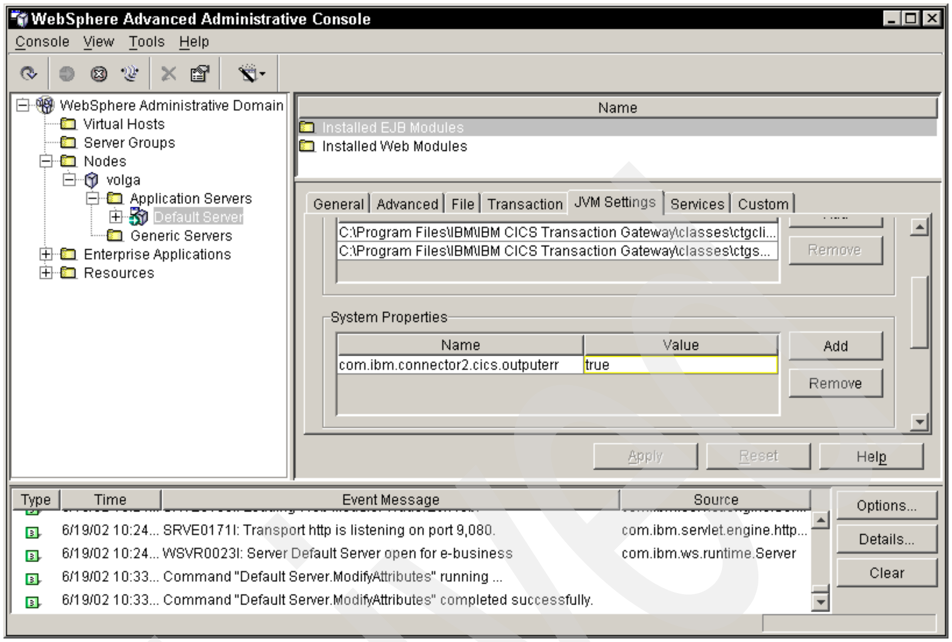


Figure 11-33 Defining the Java system properties for the default server

To activate any changes to the default server or a connection factory, the default server must be restarted. To do this, expand **WebSphere Administrative Domain -> Nodes ->** ***node name*** **-> Application Servers**. Right-click **Default Server**. Stop the server if it is running, then select **Start** to restart the default server. The connection factory trace output will now go to standard error, so it will appear in the WebSphere standard error log.

JNI trace

CICS TG JNI trace is produced when using a local CICS TG, and is controlled with the Java system property `gateway.T.setJNITFile`. The property value specifies the file name where the JNI trace is output. This must be set on the application server. To do this, perform the following steps:

1. From the WebSphere administrative console, expand **WebSphere Administrative Domain -> Nodes ->** ***node name*** **-> Application Servers**. Click **Default Server.**
2. Click the **JVM Settings** tab.
3. In the System Properties section of the JVM Settings tab, click the **Add** button. In the Name field enter `gateway.T.setJNITFile`. In the Value field enter `C:\WebSphere\AppServer\logs\ctgjni.trc` and then click **Apply**.

   The completed window should now look as shown in Figure 11-34.

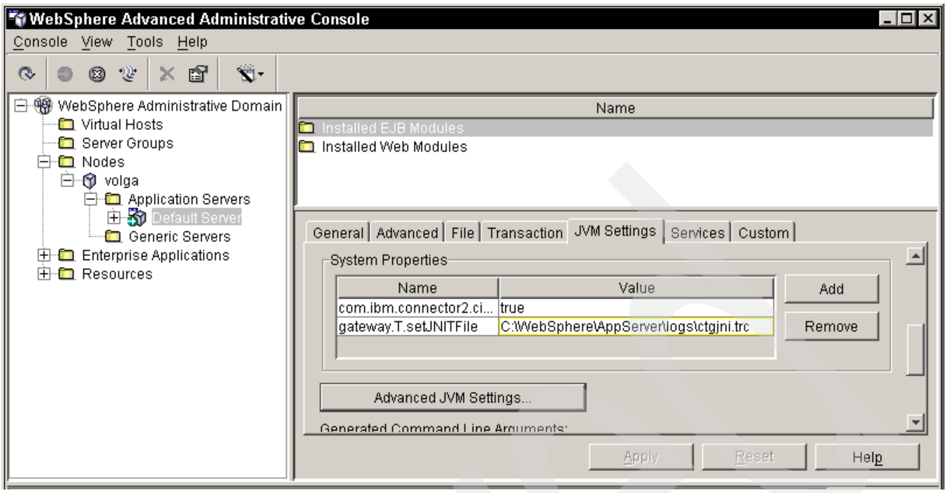


Figure 11-34 Defining the JNI trace system property

The default server must be restarted for the Java system property to be activated. The JNI trace output will now go to the file ctgjni.trc in the WebSphere logs directory.

Note: If a plain file name is specified for the JNI Java system property, the file will be created in the WebSphere bin directory, in \WebSphere\AppServer\bin.

Tip: CICS TG Java class library JNI trace is only produced when using a local Gateway. When using a remote Gateway daemon, the JNI trace must be enabled on the remote Gateway machine. For information about how to enable JNI trace on a Gateway daemon, refer to 7.4.2, “Tracing” on page 170.

Part 5

Appendixes

In this section, we describe the sample applications provided with this book, and how to install and use them.

We also provide details on how to set up and install DFHCNV data conversion templates for conversion of ASCII data to EBCDIC.

A

DFHCNV and CICS data conversion

If you chose to convert data within CICS, then this data conversion will be performed by the CICS data conversion program DFHCCNV, in combination with conversion templates defined using the DFHCNV macro. When using the CICS Transaction Gateway (CICS TG), conversion is different for External Call Interface (ECI) and External Presentation Interface (EPI) requests, so we examine each of these separately in the following sections.

Note: It is also possible to convert character data from ASCII to EBCDIC, and numeric data from little-endian to big-endian within the Java client application. This can be achieved either by using the data marshalling capabilities of the Java Record Framework, or by writing your own conversion routines. For further guidance on these options, please refer to Appendix B, “Data conversion” in the redbook, *Java Connectors for CICS*, SG24-6401.

ECI applications

ECI applications use the facilities of the CICS mirror program to link to the specified user program, passing a buffer known as the COMMAREA for input and output. The CICS mirror program (DFHMIRS) invokes the services of the data conversion program (DFHCCNV) to perform the necessary conversion of the COMMAREA (Figure A-1).

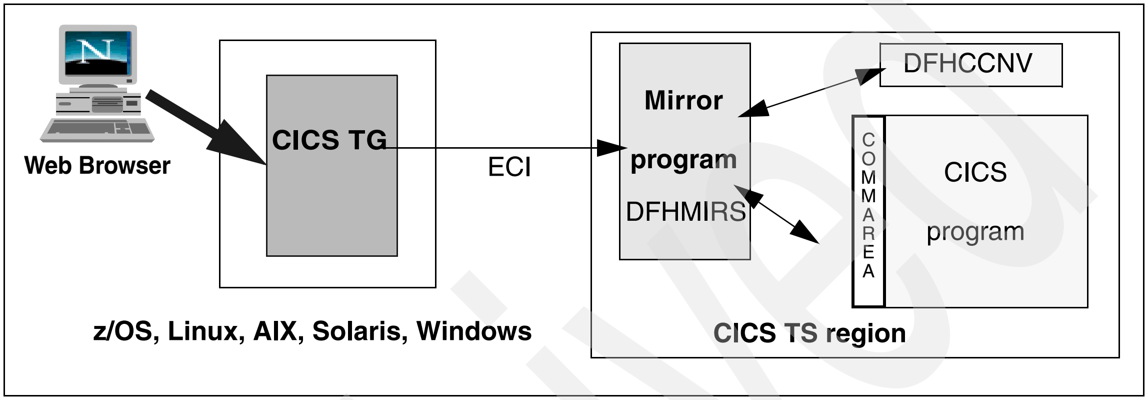


Figure A-1 CICS Transaction Gateway, ECI data conversion

Only if DFHCCNV finds a conversion template in the DFHCNV table that matches the program name will it perform code page translation for the COMMAREA associated with the ECI request. Example A-1 shows the DFHCNV table entry we used for the program ECIPROG.

Example: A-1 DFHCNV entry

```
DFHCNV  TYPE=ENTRY,RTYPE=PC,                                    *
        CLINTCP=850,SRVERCP=037,                                *
        RNAME=ECIPROG,USREXIT=NO
DFHCNV  TYPE=SELECT,OPTION=DEFAULT
DFHCNV  TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,                  *
        DATALEN=27,LAST=YES
```

The SRVERCP should represent the EBCDIC code page in which the data is stored within CICS. The CLINTCP should be the default code page for client requests, but this can be overridden by information flowed from the CICS Universal Client, which is determined from the code page of the client machine. (Note that the CICS TG supplied `CicsCpRequest` object allows you to determine the code page used by the CICS Universal Client). The DATALEN should be the maximum length of the COMMAREA you require to be translated.

The name of the mirror transaction that executes the ECI request can be modified using the extended constructor for the `ECIRequest` object (Figure A-2), but it will default to CPMI if using the CICS TG on a distributed platform or CSMI if using V4 or V5 of the CICS TG on z/OS (see Table A-1 on page 332). This can be important, since it can affect the way in which data conversion occurs.

```
byte abCommarea[] = new byte[];
abCommarea = "abcd".getBytes();
String strTranid = "MIR1"; // Private CICS mirror
JavaGateway jgaConnection = new JavaGateway();
jgaConnection.open;

ECIRequest eciRequest = new ECIRequest();
ECIRequest.ECI_SYNC_TPN, // Call type
strServerName, // CICS Server
strUserName,   // CICS userid
strPassword,   // Password
strProgramName,  // Program name
strTranid,   // Mirror eci_transid
abCommarea); // Commarea byte array

jgaConnection.flow(eciRequest);
String strCommarea = new String(abCommarea);
```

Figure A-2 Specifying a private mirror on the ECIRequest constructor

For ECI requests from distributed platforms, the mirror transaction always calls DFHCCNV, and, if a DFHCNV conversion template is present, data conversion will take place. DFHCCNV is always called, because the distributed CICS TG sets a *data conversion trigger* in the ECI call, and also flows a client code page (which will override the CLINTCP setting in the DFHCNV template). For ECI requests from the CICS TG on z/OS, this data conversion trigger is also set if the EXCI uses an EXCI Version 2 call, and so DFHCCNV is always called. However, unlike ECI requests from distributed platforms, the client code page is not flowed with the ECI request, and so the value from the CLINTCP parameter in the DFHCNV template will be used.

EXCI Version 2: In CICS TS V1.3, the fix for APAR PQ38644 adds the EXCI Version 2. In contrast to the earlier version, Version 2 supports the data conversion trigger in EXCI calls. The CICS TG on z/OS V4 and V5 exploits this trigger and DFHCCNV is therefore always called for an ECI request.

The use of EXCI Version 2 also removes the need for the DFHMIRR program, which was introduced as an optional workaround to always trigger DFHCCNV from the mirror transaction. Since it is not shipped with CICS, it must be created as a copy of the supplied mirror program DFHMIRS. If you have implemented this use of DFHMIRR, you should install the fix for APAR PQ38644 after migrating to CICS TS V1.3, then upgrade your CICS TG to V4 or V5 to enable use of the EXCI Version 2, and so abandon DFHMIRR altogether.

This technique is to be encouraged, since it also solves the possible problem of double code page conversion in a CICSplex. This can occur when the user application specified in the `ECIRequest` is actually in a region remote from the region that first runs the mirror transaction. The mirror transaction will set the data conversion trigger to *off* when routing the DPL request through to the remote region. Prior to the EXCI Version 2, you had to specify different DFHCNV tables for the two CICS regions, or associate different mirror program names to the mirror transactions in each region.

Table A-1 Mirror transaction characteristics for ECI requests

	LU 6.2 or TCP62 connection	**EXCI V2 connection (CICS TG V4 or 5)**	**EXCI Version 1 connection**
Default ECI mirror transid	CPMI	CSMI	CPMI
Supports data conversion trigger?	Yes, and the trigger is always set in the ECI request	Yes, and the trigger is always set in the ECI request	No
Mirror calls DFHCCNV on DPL request?	If data conversion trigger is set, mirror transid is CPMI, or mirror program is DFHMIRR	Only if data conversion trigger is set, mirror transid is CPMI, or mirror program is DFHMIRR	Only if mirror transid is CPMI, or mirror program is DFHMIRR

Java on z/OS

Java programs executing on z/OS and using the CICS TG ECI methods deserve a special mention because of the code page considerations on z/OS. Java strings are stored in Unicode, and Java byte arrays are stored in the native code page of the operating system. The COMMAREA flowed to CICS in an `ECIRequest` is a Java byte array, but is often converted to or from a string, for handling within the Java code. If the Java program executes in a Java Virtual Machine (JVM) on z/OS, then conversion of a string to a byte array will require the encoding to be specified.

In our previous example (Figure A-2 on page 331), we used the method `String.getBytes()` to convert our string to a byte array, and the default `String` constructor to convert the byte array to a string. This will convert the byte array to a string using the default platform encoding. On Intel or UNIX platforms, this encoding will probably be US ASCII (ISO 8859-1); however, on z/OS it will be

1047 (an extended 037 EBCDIC code page). Unicode and ASCII share the first 256 code points, so conversion works fine on ASCII platforms, but on z/OS it will not give the correct results. The solution to this is to make the Java application *code page aware*, by defining the code page of the COMMAREA byte array in the Java code (Figure A-3).

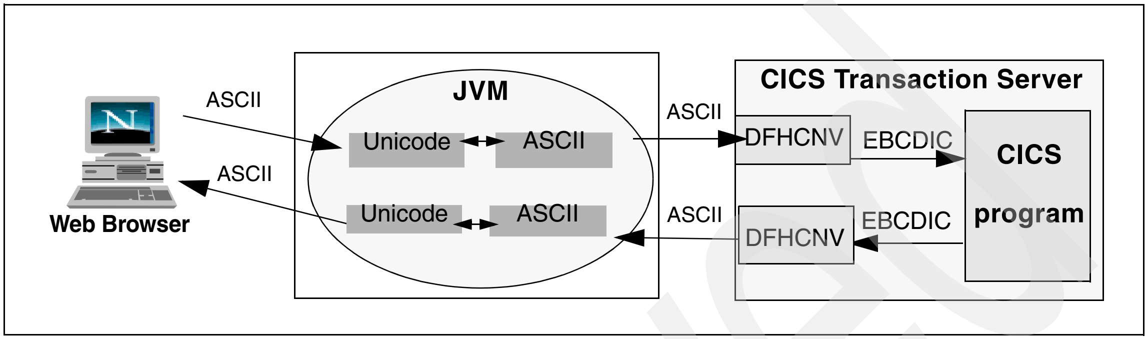


Figure A-3 Code page-aware servlet, ASCII input to CICS

Figure A-4 shows a sample of a code page-aware Java ECI application, that converts the COMMAREA to and from ASCII using the encoding parameter on the `String.getBytes()` method and the modified `String` constructor.

```
byte abCommarea[] = new byte[];
abCommarea = "abcd".getBytes("8859_1");
JavaGateway jgaConnection = new JavaGateway();
jgaConnection.open;
ECIRequest eciRequest = new ECIRequest();
ECIRequest.ECI_SYNC, // Call type
    strServerName, // CICS Server
    strUserName,   // CICS userid
    strPassword,   // Password
    strProgramName,  // Program name
    abCommarea); // Commarea byte array
jgaConnection.flow(eciRequest);
String strCommarea = new String(abCommarea,"8859_1");
```

Figure A-4 Code page-aware Java application— ASCII COMMAREA

Note: This technique means that the COMMAREA always flows to CICS in ASCII, and so you will then need to ensure that you create a correct DFHCNV table entry in your CICS region in order to convert the COMMAREA from ASCII to EBCDIC. This is not as inefficient as it sounds, since data conversion from Unicode to ASCII is an efficient operation in Java, since it just involves the removal of the high order byte.

An alternative to this is to create an EBCDIC byte array within the Java code (Figure A-5). This removes the need for the DFHCNV entry, but increases the cost within the Java code, because conversion from Unicode to EBCDIC in Java requires a table lookup. A Java code example is shown in Figure A-6.

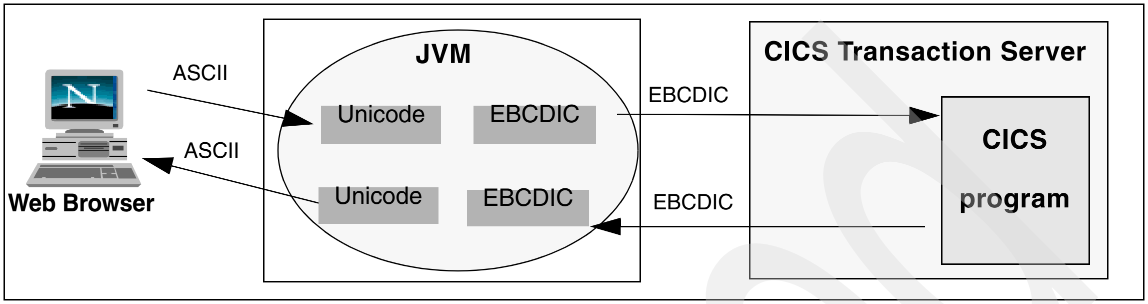


Figure A-5 Code page-aware servlet, EBCDIC input to CICS

```
byte abCommarea[] = new byte[];
abCommarea = "abcd".getBytes("037");

JavaGateway jgaConnection = new JavaGateway();
jgaConnection.open;

ECIRequest eciRequest = new ECIRequest();
ECIRequest.ECI_SYNC, // Call type
strServerName, // CICS Server
strUserName,   // CICS userid
strPassword,   // Password
strProgramName,  // Program name
abCommarea); // Commarea byte array

jgaConnection.flow(eciRequest);
String strCommarea = new String(abCommarea,"037");
```

Figure A-6 Code page-aware Java application— EBCDIC COMMAREA

EPI applications

For EPI requests from the CICS TG, the CICS data conversion program, DFHCCNV is also used to convert data from the ASCII code page of the CICS TG platform to the EBCDIC code page of the CICS region (Figure A-7).

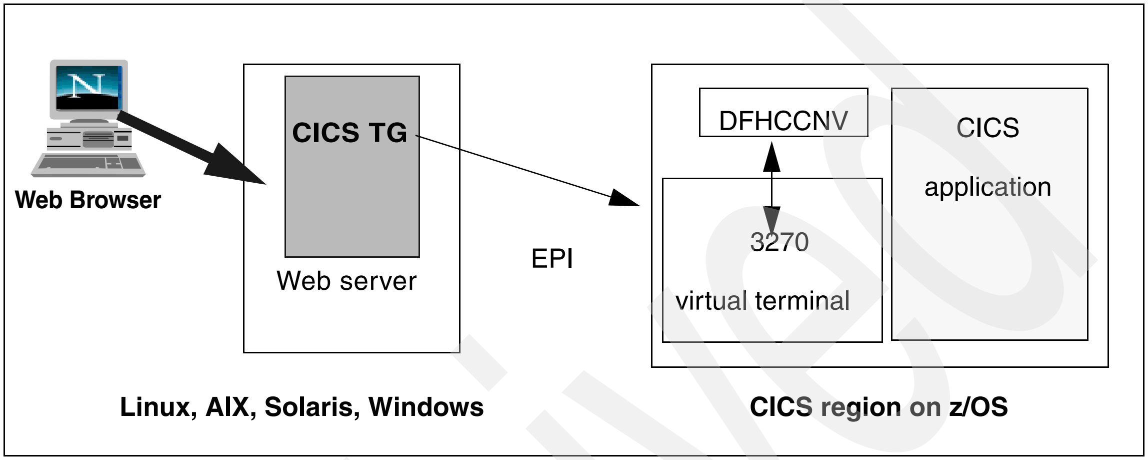


Figure A-7 CICS Transaction Gateway, EPI data conversion

All EPI requests must first create a virtual 3270 terminal definition in CICS. This definition is installed using the `EPIRequest.addTerminal()` method or the `Terminal` object. CICS then runs the transactions specified in subsequent EPI requests as if they were started on that terminal. When the terminal definition is autoinstalled in CICS, it is, like all autoinstalled terminals, created using a CICS TYPETERM definition. The CGCSGID value in the TYPETERM definition determines the server code page. The CICS TG supplies the client code page from the CICS Universal Client installation. This can be overridden using the CCSid field on the `EPIRequest.addTerminal()` method.

The available code page pairs are a subset of the those available for ECI applications. Refer to *CICS Family: Communicating from CICS on System/390®*, SC33-1697, for the most recent list.

B

Sample applications

In this appendix, we detail the two sample applications provided with the redbook:

- **CTGTesterECI**

  The CTGTesterECI application uses the `ECIRequest` class (provided in the CICS TG base classes) to flow an ECI request to a CICS region. It uses a simple JSP/servlet design with all the business logic in the servlet.

- **CTGTesterCCI**.

  CTGTesterCCI uses the J2EE CCI classes and the ECI resource adapter to flow an ECI request to CICS. As such, it will only run in an Application Server where the ECI resource adapter has already been installed. It uses a JSP/servlet/EJB design with the business logic split between the servlet and the EJB session bean.

Both of these applications are designed for the testing of connectivity from the Web application server to CICS. As such, all the CICS dependencies, such as COMMAREA size, input, encoding and length, are externalized to enable you to test different configurations. They are not designed as production applications, since such externals are unlikely to be exposed in real-life customer applications.

The CTGTesterECI application

The CTGTesterECI enterprise application contains a Java servlet that we used during this project in our various test scenarios. It is intended to provide a simple enterprise application that can be used “out of the box” to test ECI connectivity from any CICS Transaction Gateway to any configured CICS region. It is written for simplicity and thus is not intended as a real-life customer application. All input parameters can be specified as HTTP query strings or HTML form parameters, and all output is through text-based HTML.

It has the following features:

- Setting of the CICS server, mirror transaction, user ID and password
- Setting of the Gateway daemon URL and port
- Setting of the COMMAREA input, encoding, and length
- Setting of CICS TG Java client trace

The application consists of a number of JSPs and a servlet. Figure B-1 illustrates the complete architecture of the enterprise application.

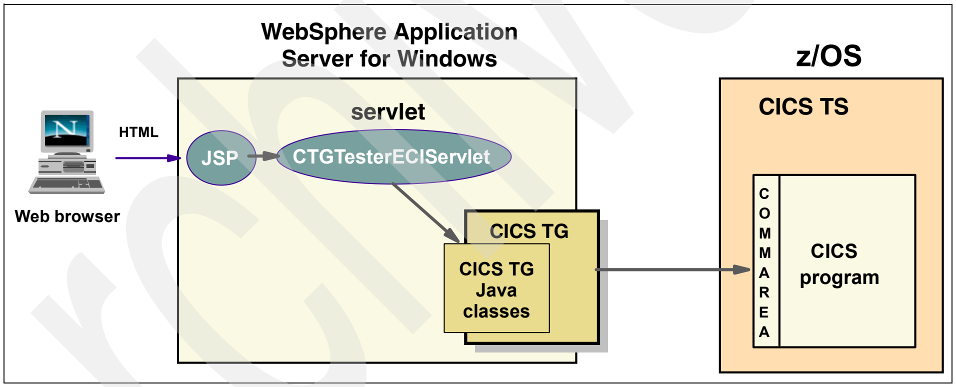


Figure B-1 Architecture of CTGTesterECI

Application overview

The following list explains the sequence of events that occur when the end user interacts with the application through a Web browser, illustrated in Figure B-2 on page 339.

1. The end user opens the Web application using a Web browser.
2. The end user clicks a button on a Web page that submits a form to the servlet.
3. The servlet receives the request for action and uses the CICS TG Java classes to execute a program on the CICS server.

4. The CICS TG Java classes return data from the CICS program to the servlet.
5. The servlet forwards to a JSP, which displays the response to the end user.

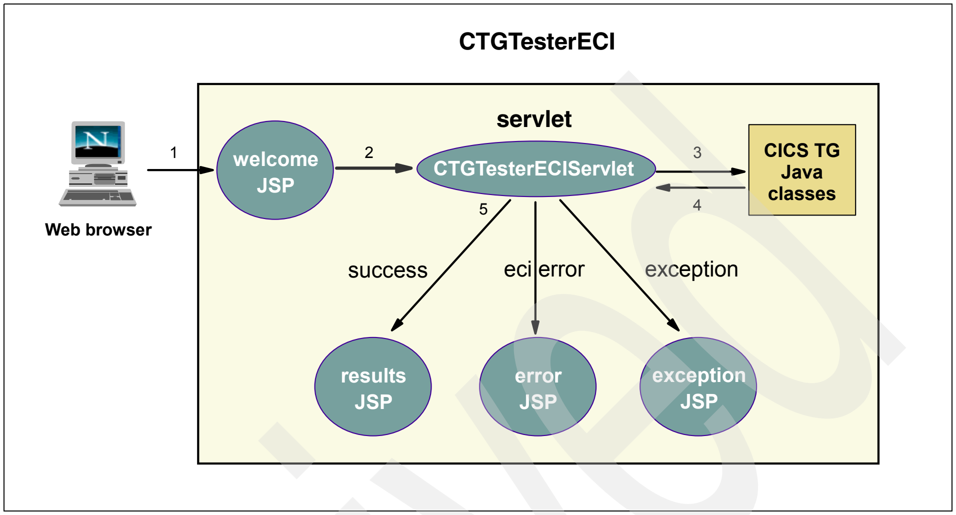


Figure B-2 Flow of CTGTesterECI

We developed and tested our application using WebSphere Studio Application Developer Integration Edition. We chose this environment instead of VisualAge for Java V4, because it supports editing of EJB 1.1 deployment descriptors, and can generate deployment code suitable for the WebSphere Application Server V4 container. It can export deployed application files directly, ready for installation into WebSphere Application Server V4. These application files include J2EE enterprise archives (EAR files), Web application archives (WAR files), and EJB 1.1 deployed JAR files. Application Developer also provides a suitable local test environment for testing your application components because it uses WebSphere Application Server Advanced Single Server V4.

As an alternative to using Application Developer, you can use VisualAge for Java in combination with the Application Assembly Tool (AAT) to produce deployable enterprise beans for WebSphere Application Server Advanced Edition.

HTML form

The HTML form is in index.jsp. It consists of a number of fields where data for the application is entered. The fields are explained in Table B-1 on page 339.

Table B-1 Fields in the HTML form, CTGTesterECI

Field	Name	Purpose
CICS program name	funcName	Program on CICS to call

Field	Name	Purpose
COMMAREA input	commareaInput	COMMAREA data
COMMAREA length	commareaLength	Length of the COMMAREA
Encoding	encoding	Encoding to convert data to before sending and after receiving
Gateway daemon URL	gatewayURL	URL of the Gateway daemon to use
Gateway daemon port	gatewayPort	Port of the Gateway daemon
CICS Server	server	Server name as defined to the Client daemon to use
User ID	username	User name to flow on the ECI request
Password	password	Password to flow on the ECI request
Mirror transaction	mirror	Transaction to use on the CICS server
Trace	trace	Whether to enable CICS TG Java client trace

The action of the HTML form is to call the servlet CTGTesterECIServlet, using the POST method to send the parameters. This means that the parameters are sent in the HTTP request body. Because an HTTP POST request is intended to affect some sort of change on the Web server, a Web browser will prompt to resubmit the data if the user clicks **Refresh** on the results page. Conversely, the GET method could be used, which sends the parameters on the HTTP request URL. Because HTTP GETs are not intended to alter anything on the Web server, the results page of such a request can be refreshed without a prompt being shown.

Servlet

In the following section, we describe the major code sections in the CTGTesterECIServlet servlet and how the servlet functions.

The servlet is defined to be in the package `itso.cics.eci.testereci`. Figure B-3 on page 341 shows the import statements used to give us access to the Java packages used in the servlet. The import of `javax.servlet` and `javax.servlet.http` give us access to the Java Servlet API set of classes; the import of `java.io`, `java.util` and `java.text` are required for utility classes used in the servlet; the import of `com.ibm.ctg.client` provides the CICS TG Java class library.

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.*;

import com.ibm.ctg.client.*;

import java.io.PrintWriter;
import java.io.IOException;
import java.util.Date;
import java.util.StringTokenizer;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
```

Figure B-3 Import statements

Figure B-4 on page 342 shows the opening code section of our servlet. We extend the `HTTPServlet` class to make our class an HTTP protocol servlet. Following this, we first declare our instance variables and objects. These are variables that we wish to share across all threads running within the servlet.

```
public class CTGTesterECIServlet extends HttpServlet
{
   // date loaded into application server
   private Date loadedDate;

   // jsp names
   private static final String jspResults = "results.jsp";
   private static final String jspError = "error.jsp";
   private static final String jspException = "exception.jsp";

   // names of attributes set in the request
   private static final String attrResultsString = "results";
   private static final String attrDefaultResultsString = "defaultResults";
   private static final String attrHexResultsString = "hexResults";
   private static final String attrErrorMessages = "errorMessages";
   private static final String attrErrorException = "errorException";
   private static final String attrLastLoaded = "lastLoaded";
   private static final String attrCicsProgram = "funcName";
   private static final String attrGatewayURL = "gatewayURL";
   private static final String attrGatewayPort = "gatewayPort";
   private static final String attrServer = "server";
   private static final String attrUsername = "username";
   private static final String attrHttpUsername = "httpusername";
   private static final String attrMirror = "mirror";
   private static final String attrCommareaLength = "commareaLength";
   private static final String attrCommarea = "commareaInput";
   private static final String attrManaged = "managed";
   private static final String attrEncoding = "encoding";
   private static final String attrReturnCode = "returnCode";
   private static final String attrReturnString = "returnString";
   private static final String attrAbendCode = "abendCode";
   private static final String attrTrace = "trace";
   private static final String attrProgress = "progress";
```

Figure B-4 Class variables

We declare the names of the JSPs we use to display the results of the program, as well as the attribute names we use to pass the results to the JSPs.

The `Servlet` interface class contains the `init()`, `destroy()`, and `service()` methods. The first of these methods we define is our `init()` method (Figure B-5 on page 343). The purpose of the `init()` method is to perform the necessary servlet initialization. It is guaranteed to be the first method to be called on any servlet instance. The servlet implemented may choose to override this method to perform custom servlet initialization. In our `init()` method, we instantiate our `loadedDate` object so as to set the time of loading and also call the superclass `init()` method to ensure the proper `HttpServlet` initialization occurs.

```
public void init(ServletConfig sc) throws ServletException
{
   // call HttpServlet init method
   super.init(sc);

   // Set time of loading
   loadedDate = new Date();
}
```

Figure B-5 init() method

The Web application server invokes the servlet `service()` method upon receiving an HTTP request targeted towards that servlet. This method in turn invokes the appropriate HTTP-specific method based on the type of request. `HttpServletRequest` is an input parameter and contains the HTTP protocol specified header information. `HttpServletResponse` is an output parameter and contains an HTTP protocol-specific header and can return HTML data to the client. In our servlet we defined a `doGet()` and `doPost()` method so the servlet can handle HTTP GET and POST requests. Both methods are identical; they call the common `processRequest()` method, which deals with GET and POST requests in the same way.

The initial section of the `processRequest()` method is shown in Figure B-6.

```
   public void processRequest(HttpServletRequest request,
HttpServletResponse response) throws IOException
   {
      JavaGateway jgaConnection = null;
      // input parameters
      String funcName = null;
      String encoding = null;
...
      String trace = "";
...
      // the jsp to forward to for displaying the results
      String jsp = jspResults;
      // values for strings to be converted into
      int commareaLengthInt = 0;
      int gatewayDaemonPort = 2006;
      // retrieve parameters
      funcName = request.getParameter("funcName");
      encoding = request.getParameter("encoding");
...
      trace = request.getParameter("trace");
```

Figure B-6 Getting HTML form input

First of all, we set the automatic variable `jsp` to the name of the JSP we will forward to at the end of servlet processing. Initially we set this to the name of the results JSP. Then we retrieve the parameters of the HTML form into automatic `String` variables.

In the next section of code, partially shown in Figure B-7, we perform some processing on these parameters. If the user ID or password is empty, then we set the corresponding variable to `null`. Later, when we pass these into the `ECIRequest` constructor, the `null` indicates that no user ID or password has been specified. We convert the integer parameters from `Strings` into `int` primitives, setting an error message if the parameters cannot be converted.

```
      // set username and password to null if empty
      if(username.equals(""))
      {
         username = null;
      }
      if(password.equals(""))
      {
         password = null;
      }
      // set last loaded date
      request.setAttribute(attrLastLoaded, loadedDate.toString());
      // set request details
      request.setAttribute(attrUsername, username);
...
      request.setAttribute(attrTrace, trace);
...
      // convert commarea length to int
      try
      {
         commareaLengthInt = Integer.parseInt(commareaLength);

         // set commarea length attribute
         request.setAttribute(attrCommareaLength,
Integer.toString(commareaLengthInt));
      }
      catch(NumberFormatException ex)
      {
         messages += "Commarea length not a number<br>";
      }

...
      // set error messages
      request.setAttribute(attrErrorMessages, messages);
```

Figure B-7 Processing HTML form input

We also use the mechanism for passing data from the servlet into our JSPs to pass the values of the input parameters to the JSPs. This is done by setting attributes in the `HTTPRequest` object associated with the request. Each attribute has a name and a `String` value, which can be retrieved in the JSP called at the end of the servlet processing.

Next we ascertain if a user ID and password are contained in the HTTP basic authentication header. This will be the case if the Web server has challenged the Web browser to enter a user ID and password. This user ID and password is flowed in every HTTP request, and encoded using the Base64 algorithm. This encoded string can be obtained by invoking the `getHeader()` method on the `HttpServletRequest` object. The encoded string must then be passed to a `StringTokenizer` object, and decoded using an appropriate decode method. We utilized a separate utility class, Base64, for this purpose. This supplies the decode and encode methods and is packaged with CTGTesterECI.

For further details on HTTP basic authentication and Base64 encoding, refer to the redbook *Securing Web Access to CICS*, SG24-5756.

The user ID and password from HTTP basic authentication are sent to the JSP for output, however they are not flowed to CICS in the `ECIRequest`. The code for doing this is included in the source, but commented out. If you are running the servlet within WebSphere Application Server V3.5 for OS/390 using the CICS TG in local mode, then it is not necessary to explicitly specify the user ID or password in the `ECIRequest` object, since the EXCI will flow the user ID of the HTTP Server thread in the `ECIRequest` if the user ID parameter in the `ECIRequest` object is `null`. If HTTP basic authentication or SSL client authentication is enabled in the HTTP Server (using `%%CLIENT%%`), then the user ID of the thread will be the client's authenticated user ID. However, we found that when running the servlet in the Web container provided by WebSphere V4 for z/OS, this behavior no longer occurred, and the user ID was now set to the user ID under which the J2EE Server instance was running.

When using WebSphere Application Server Advanced Edition for Windows, the user ID and password should be manually set in the `ECIRequest` object if you wish to flow them onto CICS.

The next section of code enables CICS TG Java client tracing if the tracing parameter is set to `on` (Figure B-8).

```
if(trace.equals("on"))
{
   T.setDebugOn(true);
}
else
{
   T.setDebugOn(false);
}
```

Figure B-8 Enabling CICS TG Java client tracing

The next section instantiates a `JavaGateway` object, using the URL and port of the Gateway as given in the input to the servlet. This automatically connects to the Gateway daemon so no `openJavaGateway()` method call is needed. Note that this, along with the other CICS TG Java method calls, is enclosed in a try catch block, so we catch any exceptions that occur.

We then convert the input COMMAREA data into a byte array using the encoding specified in the form. We then create an `ECIRequest` object using the parameters previously input and flow the request to the CICS server, as shown in Figure B-9.

```
// create the ECI Request
eciRequest =
new ECIRequest(
   ECIRequest.ECI_SYNC,
   cicsServer,
   username,
   password,
   funcName,
   mirror,
   abCommarea,
   commareaLengthInt,
   ECIRequest.ECI_NO_EXTEND,
   ECIRequest.ECI_LUW_NEW);

// indicate progess
progress += "Flowing ECI request to server<BR>";

// send the request to CICS
jgaConnection.flow(eciRequest);
```

Figure B-9 Creating an ECIRequest and flowing it to CICS

The next section of code deals with the results of the ECI request. If there have been no errors, then we convert the output COMMAREA into `String` objects, using both the default encoding and that specified in the form, and pass them to the JSP. We also convert the original byte data into a `String` hexadecimal representation, to help with data conversion debugging, and pass that to the JSP. If there has been an error, then we set `jsp` to forward to the name of the JSP that displays errors (error.jsp). In either case, we pass the ECI return codes and error strings to the JSP.

The next section of code handles any exceptions that occur while building and sending the ECI request. To deal with an exception, we build a message consisting of a timestamp and the `String` representation of the exception that occurred. We then set a request attribute containing this `String` and specify we want to forward to the JSP that displays exceptions (exception.jsp).

We then close the JavaGateway connection in a finally block, to ensure it is closed even if an exception occurs during the `flow()` method.

The final section of code in the `processRequest()` method forwards to the appropriate JSP, set in the member variable `jsp` (Figure B-10).

```
      // route to jsp
      ServletContext servletContext = getServletContext();
      RequestDispatcher dispatcher =
servletContext.getRequestDispatcher(jsp);
      try
      {
         dispatcher.forward(request, response);
      }
      catch(ServletException ex)
      {
         // log exception to servlet log
         servletContext.log("Exception routing to jsp: " + ex);
      }
```

Figure B-10 Routing to a JSP

We get the `ServletContext` object that represents the current request and response. We then get the `RequestDispatcher` object for the selected JSP from the `ServletContext`. The `forward()` method is called on the `RequestDispatcher`, passing in the `HttpRequest` and `HttpResponse`. This causes the request to be forwarded to the JSP, along with the objects representing the session. The JSP will then generate the response HTML.

The other method in our servlet is `byteArrayToHex()`, which takes a byte array and converts it to a `String` showing the array data in a hexadecimal form.

JSPs

The application uses three JSPs to format and display the results. One of three scenarios can occur:

- The request completes successfully
- An error is reported in the `ECIRequest` object
- An exception occurs inside the CICS TG Java class library

results.jsp

The JSP results.jsp processes successful completion of a request. The JSP defines the title of the HTML page and the style sheet used to format the HTML in the header. The JSP defines the JavaBeans components that the page uses to format and display the results (Figure B-11).

```
<jsp:useBean class="java.lang.String" id="results" scope="request"/>
<jsp:useBean class="java.lang.String" id="commareaInput" scope="request"/>
<jsp:useBean class="java.lang.String" id="encoding" scope="request"/>
<jsp:useBean class="java.lang.String" id="defaultResults" scope="request"/>
<jsp:useBean class="java.lang.String" id="hexResults" scope="request"/>
<jsp:useBean class="java.lang.String" id="returnCode" scope="request"/>
<jsp:useBean class="java.lang.String" id="returnString" scope="request"/>
<jsp:useBean class="java.lang.String" id="abendCode" scope="request"/>
```

Figure B-11 JavaBeans components used by results.jsp

These components are all in the request scope and are all `String` objects. Their IDs correspond to the names of the attributes we set in the `HttpRequest` in our servlet. They contain the result of the CICS request made by the servlet.

The page then includes our common JSP header, which displays the information specified in the initial form, such as Gateway daemon URL and user ID. The rest of the page then outputs the values of the request attributes using the scriptlet format `<%= JavaBean id %>`, shown in Figure B-12. In our application, the JavaBean component `returnCode` contains the ECI return code, and is inserted into the HTML output with the JSP scriplet `<%= returnCode %>`.

```
<TABLE BGCOLOR=white CELLPADDING=0><TBODY>
<TR><TH ALIGN=LEFT>Return Codes </TH></TR>
<TR><TD>ECI return code:</TD><TD BGCOLOR=yellow><%= returnCode %></TD></TR>
<TR><TD>ECI error msg:</TD><TD BGCOLOR=yellow><%= returnString %></TD></TR>
<TR><TD>Abend code:</TD><TD BGCOLOR=yellow><%= abendCode  %></TD></TR>
</TBODY></TABLE>
```

Figure B-12 Outputting the results into the HTML

The results page outputs the COMMAREA data converted to a `String` using the JVM default encoding, the encoding specified in the form and in a hexadecimal representation. The page also outputs the ECI return code, error message, and CICS abend code (which should show normal).

error.jsp

The JSP error.jsp processes the results when an ECI error occurred. The file is similar to results.jsp, except it does not display the COMMAREA output (as there is none), but only shows the ECI return code, error string, and CICS abend code, in a highlighted format. The page also shows any errors found in the processing of the form parameters by the servlet.

exception.jsp

The JSP exception.jsp is used when an exception occurs inside the CICS TG Java class library. It does not display output COMMAREA or ECI return codes, as generally there won't be either. Instead, it displays the contents of the exception that occurred and any errors found in the processing of the form parameters by the servlet.

Application XML

The XML that describes the enterprise application can be examined from within WebSphere Studio.

Web deployment descriptor

To view the Web application XML, perform the following steps:

1. From J2EE Perspective Navigator, right-click **CTGTesterECIWeb** and select **Edit Deployment Descriptor**.

This displays the Web deployment descriptor editor's General tab. Click the **Servlets** tab to show the Servlets pane. Click **CTGTesterECIServlet** in the Servlets list (Figure B-13 on page 350).

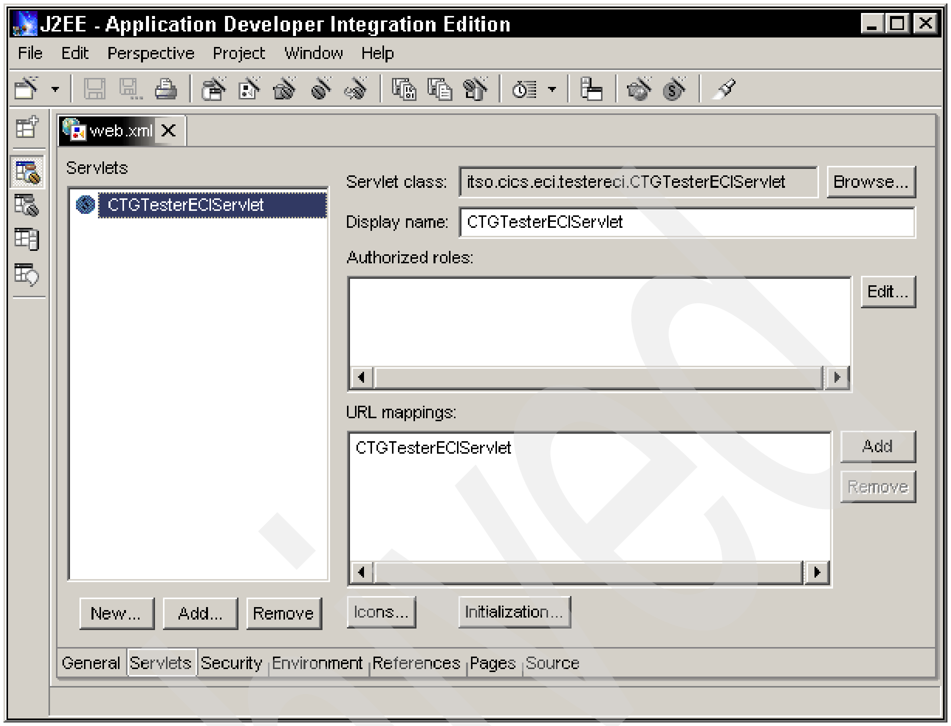


Figure B-13 CTGTesterECI Web deployment descriptor servlets panel

We can see that the Servlet class is `itso.cics.eci.testereci.CTGTesterECIServlet`, the class we have outlined in the previous section. The display name is `CTGTesterECIServlet`. The URL mappings section has one entry: `CTGTesterECIServlet`. This means that a URL of `CTGTesterECIServlet` relative to the Web application root will map to this servlet. Because of the Servlet class entry, this URL will invoke the class `itso.cics.eci.testereci.CTGTesterECIServlet`.

There is nothing of note on the Security, Environment or References tabs.

Click the **Pages** tab to show the Pages pane, shown in Figure B-14 on page 351. Here we can see the entry `index.jsp` in the Welcome files pane. This means that a URL that specifies the Web application root, for example “/”, will cause the page index.jsp to be looked for in the Web application. We supply an index.jsp, so this page will be found and displayed in the Web browser.

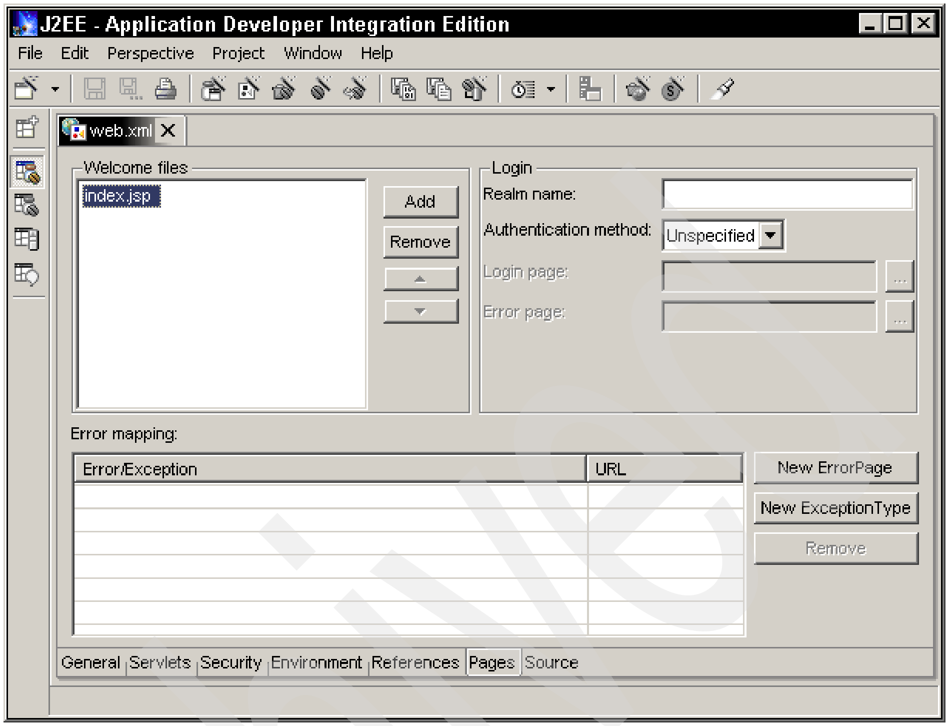


Figure B-14 CTGTesterECI Web deployment descriptor pages panel

Click the **Source** tab to display the Source pane. This shows the source of the web.xml file that is generated from entries in the Web deployment descriptor editor (Figure B-15 on page 352).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app id="WebApp">
   <display-name>CTGTesterECIWeb</display-name>
   <servlet>
      <servlet-name>CTGTesterECIServlet</servlet-name>
      <display-name>CTGTesterECIServlet</display-name>

<servlet-class>itso.cics.eci.testereci.CTGTesterECIServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>CTGTesterECIServlet</servlet-name>
      <url-pattern>CTGTesterECIServlet</url-pattern>
   </servlet-mapping>
   <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
   </welcome-file-list>
</web-app>
```

Figure B-15 CTGTesterECI Web deployment descriptor source

As can be seen, the display name of our Web application is specified with the <display-name> tag. Most of the values shown in the XML editor are marked up using the same name, except for the servlet mapping, which is marked up with the <servlet-name> tag to show which servlet the <url-pattern> corresponds to.

Web context root

The context root of our Web application can be viewed and edited from the Web pane in the enterprise application properties window in WebSphere Studio. To see this window, perform the following steps:

1. From the J2EE Perspective Navigator view, right-click **CTGTesterECIWeb** and select **Properties**.
2. From the Properties dialog tree, click the **Web** node. This shows the Web panel (Figure B-16 on page 353).

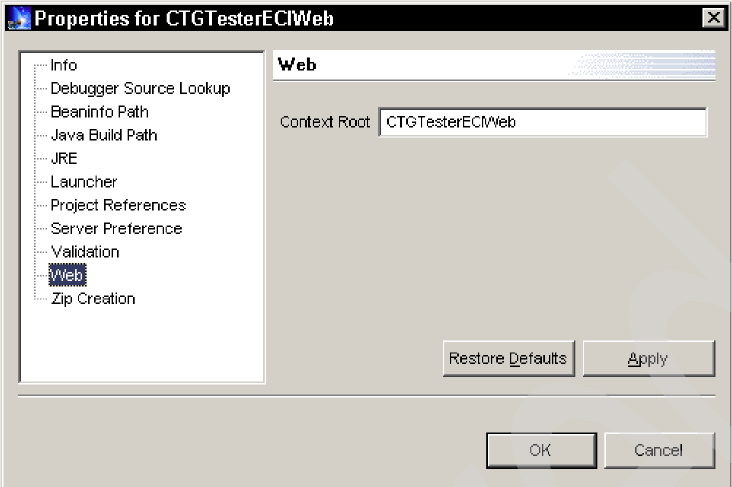


Figure B-16 CTGTesterECIWeb properties Web panel

As shown in the Context Root field, the context root for our Web application is CTGTesterECIWeb. Therefore, the URL to invoke the application is `/CTGTesterECIWeb`, assuming that, at deploy time, the Application Server is set to map enterprise applications directly to /. Because the servlet URL mapping is relative to the context root, the URL `/CTGTesterECIWeb/CTGTesterECIServlet` can be used to directly invoke the servlet.

Application deployment descriptor

To view the application deployment descriptor, perform the following steps from WebSphere Studio:

1. From the J2EE Perspective Navigator view, expand **CTGTesterECI -> META-INF**.
2. Right-click **application.xml** and select **Open**.

The General tab shows some information about the application. Click the **Modules** tab to show the Modules pane. The Modules pane defines which modules make up the enterprise application and shows which URL invokes the application. Click **CTGTesterECIWeb.war** (Figure B-17 on page 354).

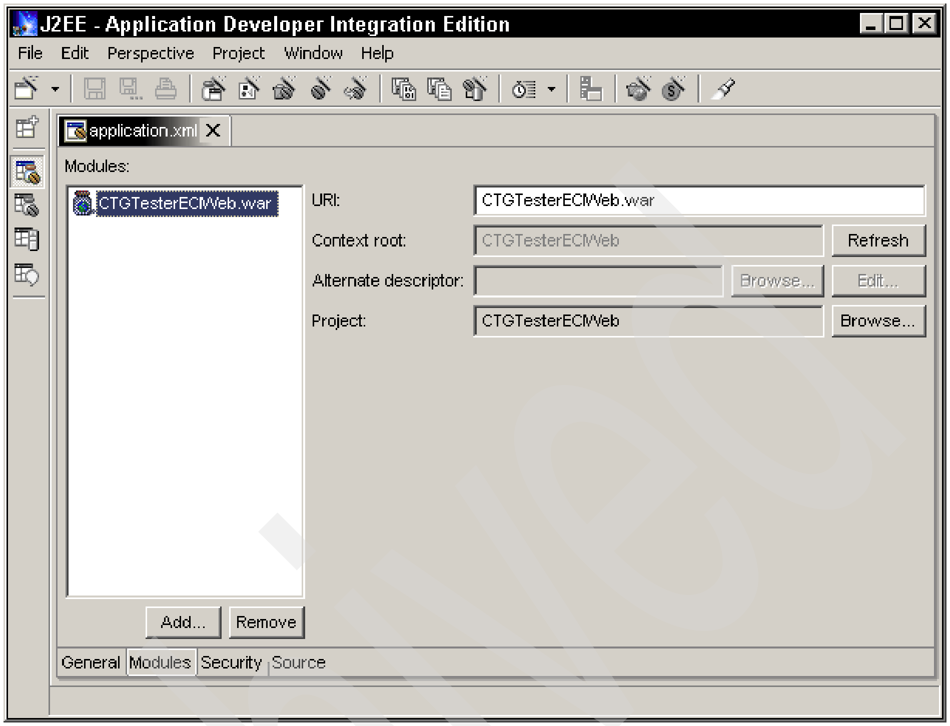


Figure B-17 CTGTesterECI application.xml modules panel

As shown, the application uses the CTGTesterECIWeb module. This is a Web application, so it is contained in a Web Archive Resource (WAR) file. When CTGTesterECIWeb is built, WebSphere Studio builds it into a WAR file with the name of the Web application, and makes it available to the CTGTesterECI application. The URI of this WAR file is therefore `CTGTesterECIWeb.war`. The Context root field is set to `CTGTesterECIWeb`, from the CTGTesterECIWeb application properties.

The CTGTesterCCI application

The CTGTesterCCI enterprise application contains a Java servlet and a session bean that we used during this project in our various test scenarios. It is intended to provide a simple enterprise application that can be used “out of the box” to test ECI connectivity from any CICS Transaction Gateway to any configured CICS region using the CICS ECI resource adapter. It is written for simplicity and thus is not intended as a real-life customer application. All input parameters can be specified as HTTP query strings or HTML form parameters, and all output is through text-based HTML.

It has the following features:

- Setting of the COMMAREA input, encoding, and length
- Setting of CICS TG Java client and CCI trace
- Executing a program a number of times
- For an unmanaged connection, setting of the CICS application server, CICS mirror transaction, CICS user ID, and password
- For an unmanaged connection, setting of the Gateway daemon URL and port

The application consists of a number of JSPs, a servlet, and an enterprise bean. Figure B-1 on page 338 illustrates the complete architecture of the enterprise application.

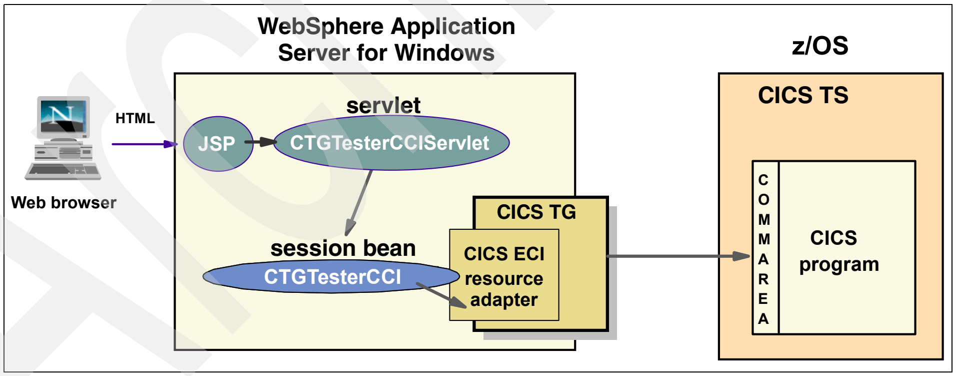


Figure B-18 Architecture of CTGTesterCCI

Application overview

The following is the sequence of events that occur when the end user interacts with the application through a Web browser, illustrated in Figure B-19:

1. The end user opens the Web application using a Web browser.
2. The end user clicks a button on a Web page that submits a form to the servlet.
3. The servlet receives the request for action and calls a method on the remote interface of the CTGTesterCCI session bean.
4. The session bean uses the CICS ECI resource adapter to call the CICS program.
5. The CICS ECI resource adapter returns data from the CICS program to the session bean.
6. The session bean returns the output data from the CICS program back to the servlet.
7. The servlet forwards to a JSP, which displays the response to the end user.

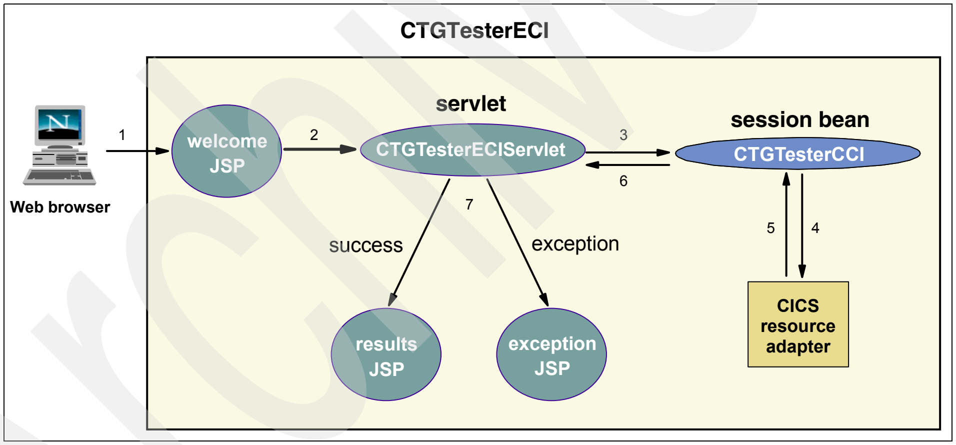


Figure B-19 Flow of CTGTesterCCI

We developed and tested our application using WebSphere Studio Application Developer Integration Edition. We chose this environment instead of VisualAge for Java V4, because it supports editing of EJB 1.1 deployment descriptors, and can generate deployment code suitable for the WebSphere Application Server V4 container. It can export deployed application files directly, ready for installation into WebSphere Application Server V4. These application files include J2EE enterprise archives (EAR files), Web application archives (WAR files), and EJB 1.1 deployed JAR files. Application Developer also provides a suitable local test environment for testing your application components because it uses WebSphere Application Server Advanced Single Server V4.

As an alternative to using Application Developer, you can use VisualAge for Java in combination with the Application Assembly Tool (AAT) to produce deployable enterprise beans for WebSphere Application Server Advanced Edition.

HTML form

The HTML form is in index.jsp in the CTGTesterCCIWeb enterprise application project in WebSphere Studio. It is stored under the webApplication folder. The HTML form consists of a number of fields where data for the application is entered. Some of the fields are not used if a managed connection factory is being used. These are marked as *unmanaged options* in the HTML. The fields on the form are shown in Table B-2.

Table B-2 Fields in the HTML form, CTGTesterCCI

Field	Name	Purpose
Managed	managed	Whether to use a managed connection factory or instantiate one directly
CICS program name	funcName	Program on CICS to call
COMMAREA input	commareaInput	COMMAREA data
COMMAREA length	commareaLength	Length of the COMMAREA
Encoding	encoding	Encoding to convert data to before sending and after receiving
Iterations	iterations	The number of times to run the CICS program
Application trace	appTrace	Whether to enable CICS TG Java client trace
Unmanaged options		
Gateway daemon URL	gatewayURL	URL of the Gateway daemon to use
Gateway daemon port	gatewayPort	Port of the Gateway daemon
CICS Server	server	Server name as defined to the Client daemon to use
User ID	username	Username to flow on the ECI request
Password	password	Password to flow on the ECI request
Mirror transaction	mirror	Transaction to use on the CICS server
CCI Trace level	trace	The level of CCI trace

The action of the HTML form is to call the servlet CTGTesterCCIServlet, using the POST method to send the parameters. This means that the parameters are sent in the HTTP request body. Because an HTTP POST request is intended to affect some sort of change on the Web server, a Web browser will prompt to resubmit the data if the user clicks **Refresh** on the results page. Conversely, the GET method could be used, which sends the parameters on the HTTP request URL. Because HTTP GETs are not intended to alter anything on the Web server, the results page of such a request can be refreshed without a prompt being shown.

Servlet

In the following section, we describe the major code sections in the CTGTesterCCIServlet servlet and how the servlet functions. The servlet is stored in the CTGTesterCCIWeb enterprise application project in WebSphere Studio.

The servlet is defined to be in the package `itso.cics.j2ee.eci.testercci`. It is stored in the folder source/itso/cics/j2ee/eci/testercci in WebSphere Studio. Figure B-20 shows the import statements used to give us access to the Java packages used in the servlet. The import of `javax.servlet` and `javax.servlet.http` give us access to the Java Servlet API set of classes; the import of `javax.naming` gives us access to the JNDI library; the import of `javax.rmi` gives us access to the Java Remote Method Invocation library; the imports of `java.io`, `java.util`, and `java.text` are required for utility classes used in the servlet; the import of `com.ibm.ctg.client` provides the CICS TG Java class library.

```
import javax.servlet.http.*;
import javax.servlet.*;
import javax.naming.*;
import javax.rmi.*;

import java.util.Date;
import java.util.StringTokenizer;
import java.io.UnsupportedEncodingException;
import java.io.IOException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
```

Figure B-20 Import statements

Figure B-21 shows the opening code section of our servlet. We extend the `HTTPServlet` class to make our class an HTTP protocol servlet. Following this we then first declare our instance variables and objects, which are variables that we wish to share across all threads running within the servlet.

```
public class CTGTesterCCIServlet extends HttpServlet
{

   // date loaded into application server
   private Date loadedDate;

   // jsp names
   private static final String jspResults = "results.jsp";
   private static final String jspError = "error.jsp";

   // names of attributes set in the request
   private static final String attrResultsString = "results";
   private static final String attrDefaultResultsString = "defaultResults";
   private static final String attrHexResultsString = "hexResults";
   private static final String attrErrorException = "errorException";
   private static final String attrErrorMessages = "errorMessages";
   private static final String attrLastLoaded = "lastLoaded";
   private static final String attrCicsProgram = "funcName";
   private static final String attrGatewayURL = "gatewayURL";
   private static final String attrGatewayPort = "gatewayPort";
   private static final String attrServer = "server";
   private static final String attrUsername = "username";
   private static final String attrHttpUsername = "httpusername";
   private static final String attrMirror = "mirror";
   private static final String attrCommareaLength = "commareaLength";
   private static final String attrCommarea = "commareaInput";
   private static final String attrManaged = "managed";
   private static final String attrEncoding = "encoding";
   private static final String attrTrace = "trace";
   private static final String attrIterations = "iterations";
   private static final String attrAppTrace = "appTrace";
```

Figure B-21 Class variables

We declare the names of the JSPs we use to display the results of the program, as well as the attribute names we use to pass the results to the JSPs.

The Servlet interface class contains the `init()`, `destroy()`, and `service()` methods. The first of these methods we define is our `init()` method (Figure B-22 on page 360). The purpose of the `init()` method is to perform the necessary servlet initialization. It is guaranteed to be the first method to be called on any servlet instance. The servlet implemented may choose to override this method to perform custom servlet initialization. In our `init()` method, we instantiate our `loadedDate` object so as to set the time of loading and also call the superclass `init()` method to ensure the proper `HttpServlet` initialization occurs.

```
public void init(ServletConfig sc) throws ServletException
{
   // call HttpServlet init method
   super.init(sc);

   // Set time of loading
   loadedDate = new Date();
}
```

Figure B-22 init() method

The Web application server invokes the servlet `service()` method upon receiving an HTTP request targeted towards that servlet. This method in turn invokes the appropriate HTTP-specific method based on the type of request. `HttpServletRequest` is an input parameter and contains the HTTP protocol specified header information. `HttpServletResponse` is an output parameter and contains an HTTP protocol-specific header and can return HTML data to the client. In our servlet we defined a `doGet()` and `doPost()` method so the servlet can handle HTTP GET and POST requests. Both methods are identical; they call the common `processRequest()` method, which deals with GET and POST requests in the same way.

The initial section of the `processRequest()` method is shown in Figure B-23 on page 361.

```
   public void processRequest(HttpServletRequest request,
HttpServletResponse response) throws IOException, ServletException
      {
         // input parameters
         String funcName = null;
         String encoding = null;
...
         String appTrace = null;
...
         // output messages buffer
         String messages = "";

         // the jsp to forward to for displaying the results
         String jsp = jspResults;

         // values for strings to be converted into
         int commareaLengthInt = 0;
...
         boolean appTraceBool = false;

         // retrieve HTTP request parameters
         funcName = request.getParameter("funcName");
         encoding = request.getParameter("encoding");
...
         appTrace = request.getParameter("appTrace");
```

Figure B-23 Getting HTML form input

First of all we set the automatic variable `jsp` to the name of the JSP we will forward to at the end of servlet processing. Initially we set this to the name of the results JSP. Then we retrieve the common parameters of the HTML form into automatic `String` variables.

In the next section of code, partially shown in Figure B-24 on page 362, we perform some processing on these parameters. We convert the integer parameters from `Strings` into `int` primitives, setting an error message if the parameters cannot be converted. We convert the parameters with a yes or no selection into `boolean` primitives. We also retrieve the parameters for an unmanaged connection if the `managed` parameter is set to `no`.

```
          // convert managed type
          if(managed.equals("yes"))
          {
             managedBool = true;
          }

          // convert app trace type
          if(appTrace.equals("on"))
          {
             appTraceBool = true;
          }

          // set common attributes in the request
          request.setAttribute(attrCicsProgram, funcName);
...
          request.setAttribute(attrAppTrace, appTrace);

          // set last loaded date in the request
          request.setAttribute(attrLastLoaded, loadedDate.toString());

          // get unmanaged parameters if needed
          if(!managedBool)
          {
             username = request.getParameter("username");
...
             trace = request.getParameter("trace");

             // set unmanaged request details
             request.setAttribute(attrUsername, username);
...
          }
...
          // set error messages in the request
          request.setAttribute(attrErrorMessages, messages);
```

Figure B-24 Processing HTML form input and getting unmanaged input

We also use the mechanism for passing data from the servlet into our JSPs to pass the values of the input parameters to the JSPs. This is done by setting attributes in the `HTTPRequest` object associated with the request. Each attribute has a name and a `String` value, which can be retrieved in the JSP called at the end of the servlet processing.

Next we ascertain if a user ID and password are contained in the HTTP basic authentication Header. This will be the case if the Web server has challenged the Web browser to enter a user ID and password. This user ID and password is flowed in every HTTP request, and encoded using the Base64 algorithm. This encoded string can be obtained by invoking the `getHeader()` method on the `HttpServletRequest` object. The encoded string must then be passed to a `StringTokenizer` object, and decoded using an appropriate decode method. We utilized a separate utility class, Base64, for this purpose. This supplies the decode and encode methods and is packaged with CTGTesterCCI.

For further details on HTTP basic authentication and Base64 encoding, refer to the redbook *Securing Web Access to CICS*, SG24-5756.

The user ID and password from HTTP basic authentication are sent to the JSP for output. However, they are not flowed to CICS. The code for doing this if using an unmanaged connection factory is included in the source, but commented out.

The next section of code performs a JNDI lookup for our session bean, shown in Figure B-25.

```
// lookup our session bean
Context ic = new InitialContext();
Object or = ic.lookup("java:comp/env/ejb/CTGTesterCCI");
CTGTesterCCI tester = null;
if (or != null)
{
   CTGTesterCCIHome home = (CTGTesterCCIHome)PortableRemoteObject.narrow(or,
CTGTesterCCIHome.class);
   tester = home.create();
}
```

Figure B-25 JNDI lookup and remote method invocation

We get a JNDI `InitialContext` and then perform a lookup using the JNDI name `java:comp/env/ejb/CTGTesterCCI`. This is a local JNDI name that will be mapped to the actual JNDI name of the session bean at deploy time. To indicate we want such a mapping, we create an EJB reference in the Web deployment descriptor with the name `ejb/CTGTesterCCI`.

Once we have looked up the object, we get the home interface of our session bean using the `narrow()` method call of the RMI class `PortableRemoteObject`. This method performs any operations necessary to allow us to invoke the bean's `create()` method and returns a reference to the session bean home interface. We then call the `create()` method on this home interface to get a reference to the remote interface we will use to invoke our business logic.

Next, we execute our business logic on the session bean and collect the results, as shown in Figure B-26 on page 364.

```
ResultsBean resultsB = tester.execute(funcName, encoding, commareaInput,
commareaLengthInt, username, password, managedBool, gatewayURL,
gatewayDaemonPort, cicsServer, mirror, traceInt, iterationsInt,
appTraceBool);

String results = resultsB.getResultsString();

// set result string attribute
request.setAttribute(attrResultsString, results);

// get byte data from results
byte[] data = resultsB.getResultsBytes();

// set result string in default encoding attribute
request.setAttribute(attrDefaultResultsString, new String(data));

// set request attribute for hex data
request.setAttribute(attrHexResultsString, byteArrayToHex(data));
```

Figure B-26 Executing the session bean business logic

The `execute()` method takes all of the HTML form parameters as input and returns a `ResultsBean` object. This is a data bean class that encapsulates a `String` object and a `byte` array, and provides methods to get and set both. It is shown in Figure B-27 on page 365.

If the session bean does not successfully complete the CICS request, then it will throw an exception, so the `execute()` method call is inside a try catch block. Assuming the `execute()` method ran successfully, we set a request attribute for the COMMAREA converted into a `String` in the encoding specified in the HTML form. We also set an attribute with the hexadecimal representation of the COMMAREA, to aid data conversion debugging, and an attribute with the COMMAREA converted into a `String` using the JVM default encoding.

```
package itso.cics.eci.j2ee.testercci;

import java.io.Serializable;

/**
 * Data Bean which stores a results string and a byte array representing the
data which created the string.
 * Initially has an empty string and zero length byte array.
 */
public class ResultsBean implements Serializable
{
   private String resultsString = "";
   private byte[] resultsBytes = new byte[0];

   public void setResultsString(String data)
   {
      resultsString = data;
   }

   public String getResultsString()
   {
      return resultsString;
   }

   public void setResultsBytes(byte[] data)
   {
      resultsBytes = data;
   }

   public byte[] getResultsBytes()
   {
      return resultsBytes;
   }
}
```

Figure B-27 ResultsBean class

The next section of code handles any exceptions that occur in the `execute()` method and is shown in Figure B-28 on page 366. To deal with an exception, we build a message consisting of a timestamp of when it was caught, and the `String` representation of the exception that occurred. We then set a request attribute containing this `String` and specify we want to forward to the JSP that displays exceptions (exception.jsp).

```
catch (Throwable t)
{
   // exception occured, set exception message in request with timestamp
   String nowStr = "";
   try
   {
      SimpleDateFormat sdf = (SimpleDateFormat)DateFormat.getTimeInstance();
      sdf.applyPattern("MM/dd/yy HH:mm:ss:SSS");
      nowStr = sdf.format(new Date());
   }
   catch(ClassCastException ex2)
   {
      // no SimpleDateFormat instance for this locale

      // format with Date toString
      nowStr = (new Date()).toString();
   }
   request.setAttribute(attrErrorException, nowStr + " : " + t);

   // change jsp to use to the error jsp
   jsp = jspError;
}
```

Figure B-28 Handling exceptions in the execute() method

The final section of code in the `processRequest()` method forwards to the appropriate JSP, set in the instance variable `jsp` (Figure B-29).

```
finally
{
   // route to jsp
   ServletContext servletContext = getServletContext();
   RequestDispatcher dispatcher = servletContext.getRequestDispatcher(jsp);
   dispatcher.forward(request, response);
}
```

Figure B-29 Routing to a JSP

We get the `ServletContext` object that represents the current request and response. We then get the `RequestDispatcher` object for the selected JSP from the `ServletContext`. The `forward()` method is called on the `RequestDispatcher`, passing in the `HttpRequest` and `HttpResponse`. This causes the request to be forwarded to the JSP, with the objects representing details of the session. The JSP will then generate the response HTML.

The other method in our servlet is byteArrayToHex(), which takes a byte array and converts it to a String showing the array data in a hexadecimal form.

Session bean

In the following section, we describe the major code sections in the CTGTesterCCIBean session bean implementation class and how the bean functions. The session bean is stored in the CTGTesterCCIEJB enterprise application package in WebSphere Studio.

The bean consists of three Java classes that define the Home interface (CTGTesterCCIHome), the Remote interface (CTGTesterCCI) and the bean implementation (CTGTesterCCIBean). A bean's home interface defines methods for locating, creating, and removing instances of beans. A bean's remote interface provides the client's view of the bean, listing the business methods available on the enterprise bean. The bean implementation provides the implementation of the business logic, and other methods required by the type of enterprise bean being created.

There are also some EJB stubs and ties in the enterprise application that are created by the RMI compiler using the menu **Generate -> Deploy and RMIC code** in WebSphere Studio. These classes are necessary to allow deployment into WebSphere Application Server and testing in the local WebSphere Studio test environment.

The bean is defined to be in the package itso.cics.j2ee.eci.testercci. It is stored in the folder ejbModule/itso/cics/j2ee/eci/testercci in WebSphere Studio. The bean implementation is in CTGTesterCCIBean, which we now examine. Figure B-30 shows the import statements used to give us access to the Java packages used in the bean. The import of com.ibm.connector2.cics give us access to the CICS resource adapter classes; javax.resource.cci gives us access to the Common Connector Interface library; the import of javax.resource gives us access to the Java resource library; the javax.ejb gives us access to the EJB classes; javax.naming.InitialContext is the JNDI naming context object needed to perform a lookup of a connection factory; the import of com.ibm.ctg.client.T provides the CICS TG Java client trace class.

```
import com.ibm.connector2.cics.*;
import com.ibm.ctg.client.T;
import javax.resource.cci.*;
import javax.resource.*;
import javax.naming.InitialContext;
import javax.ejb.*;
import java.io.PrintWriter;
```

Figure B-30 Import statements

Figure B-31 shows the opening code section of our bean. We implement the SessionBean interface to make our class a session bean. Following this, we first declare our instance variables and objects, which are variables that we wish to share across all threads running within the bean.

```
public class CTGTesterCCIBean implements SessionBean
{
    private SessionContext mySessionCtx;
    transient private Connection eciConn;
    transient private Interaction eciInt;
    transient private ECIInteractionSpec eSpec;
```

Figure B-31 Class variables

We declare a SessionContext for use in the session bean management methods. We declare a Connection, Interaction and ECIInteractionSpec for use with the CICS ECI resource adapter.

The SessionBean interface class contains the setSessionContext(), ejbActivate(), ejbPassivate() and ejbRemove() methods. We provide empty implementations of ejbActivate() and ejbPassivate(). We store a reference to the SessionContext in setSessionContext(). We define a method ejbCreate(), which is called when the bean is created, shown in Figure B-32. In this method we create an ECIInteractionSpec object, which we use when executing the CICS request.

```
public void ejbCreate() throws CreateException
{
  // create an Interaction spec to work with, This is specific to ECI
  eSpec = new ECIInteractionSpec();
}
```

Figure B-32 ejbCreate() method

The business logic of our session bean is contained in the method execute(). This is called by clients and takes details about the ECI request to execute as parameters. The initial section of the execute() method is shown in Figure B-33 on page 369.

```
public ResultsBean execute(
    String funcName,
    String encoding,
    String commarea,
    int commareaLength,
    String username,
    String password,
    boolean managed,
    String gatewayURL,
    int gatewayPort,
    String cicsServer,
    String mirror,
    int trace,
    int iterations,
    boolean appTrace)
    throws ResourceException, Exception
{
    JavaStringRecord jsr = null;
```

Figure B-33 Initial part of execute() method

We create a `JavaStringRecord` object here. This is based upon a sample shipped with the CICS TG, modified to allow access to the bytes that make up the COMMAREA data. We added the method `getData()` to `JavaStringRecord` to return the bytes that make up the COMMAREA data.

The next piece of code in Figure B-34 on page 370 sets the instance variables `eciConn` and `eciInt`, a `Connection` and `Interaction`, from either a managed or unmanaged connection factory, depending on the value of `managed`. This is done by the `getConnection()` method for a managed connection, and the `getNonManagedConnection()` method for unmanaged.

```
if (managed)
{
  getConnection();
}
else
{
  getNonManagedConnection(
     username,
     password,
     gatewayURL,
     gatewayPort,
     cicsServer,
     mirror,
     trace);
}
```

Figure B-34 getting the connection factory

After this, we enable CICS TG Java client trace if the `appTrace` parameter is true.

The next section of code sets up the `ECIInteractionSpec` object `eSpec`, shown in Figure B-35. This encapsulates the CICS program name, COMMAREA length, and details of how to flow the request. We specify `SYNC_SEND_RECEIVE` for a synchronous request.

```
// setup the interactionSpec.
eSpec.setFunctionName(funcName);
eSpec.setCommareaLength(commareaLength);
// set reply length to same size as commarea
eSpec.setReplyLength(commareaLength);
eSpec.setInteractionVerb(ECIInteractionSpec.SYNC_SEND_RECEIVE);
```

Figure B-35 Setting the ECI interaction spec

We then execute the CICS program a number of times, specified with the `iterations` parameter. The section of code for one execution is shown in Figure B-36 on page 371.

```
// create a record for use
jsr = new JavaStringRecord(encoding);
// set input data if we have any
if (commareaLength > 0)
{
  jsr.setText(commarea);
}

// make the call
try
{
  eciInt.execute(eSpec, jsr, jsr);
}
catch (ResourceException e)
{
  // output the stacktrace and re-throw it back to the client
  e.printStackTrace();
  dropConnection();
  throw e;
}
```

Figure B-36 Executing an ECI request

We create a `JavaStringRecord` object to encapsulate the COMMAREA data, both input and output, and set our input data. We then use the `Interaction` we got from the connection factory to execute the request by calling its `execute()` method. We pass the `ECIInteractionSpec` object `eSpec`, and the `JavaStringRecord` object `jsr` as input and output. If an exception occurs in the `Interaction execute()` method, we catch it, print the stack trace to the application server, drop our CICS connection, and rethrow the exception back to the remote client.

Finally, we close our CICS connection, and create and set our response `ResultsBean` object, as shown in Figure B-37. We use the text and the byte array from the `JavaStringRecord` and return the `ResultsBean`.

```
dropConnection();

// return the commarea from the last executed program
ResultsBean resultsB = new ResultsBean();
resultsB.setResultsString(jsr.getText());
resultsB.setResultsBytes(jsr.getData());

return resultsB;
```

Figure B-37 Returning the results

The `getConnection()` method of our session bean gets a managed connection to CICS using a connection factory published in the JNDI directory. The first section of the code is shown in Figure B-38.

```
private void getConnection() throws Exception
{
  ConnectionFactory cf = null;
  try
  {
     javax.naming.Context ic = new InitialContext();
     cf = (ConnectionFactory) ic.lookup("java:comp/env/ECI");
  }
  catch (Exception e)
  {
     e.printStackTrace();
     throw new Exception(
      "Lookup for java:comp/env/ECI failed. Exception: " + e.toString());
  }
  if (cf == null)
  {
     throw new Exception("Lookup for java:comp/env/ECI resulted in a null
reference");
  }
```

Figure B-38 Getting a managed CICS connection

We create an `InitialContext` object and use it to look up a `ConnectionFactory` using the JNDI name `java:comp/env/ECI`. This is a local JNDI name that will be mapped to the actual JNDI name of the connection factory at deploy time. To indicate we want such a mapping, we create a resource reference in the EJB deployment descriptor with the name `ECI`.

If an exception occurs, we print the stack trace to the application server and throw a new exception detailing what happened, along with the original exception. If we get a `null` back from the lookup, we throw an exception indicating what happened.

The next section of code sets the instance variables that reference a `Connection` and `Interaction`, shown in Figure B-39 on page 373.

```
try
{
  eciConn = (Connection) cf.getConnection();
}
catch (Exception e)
{
  e.printStackTrace();
  throw new Exception(
    "failed to get connection from ECI Factory. Exception: "
    + e.toString());
}

try
{
  eciInt = (Interaction) eciConn.createInteraction();
}
catch (Exception e)
{
  e.printStackTrace();
  throw new Exception(
    "failed to get interaction from ECI Connection. Exception: "
    + e.toString());
}
```

Figure B-39 Getting a Connection and Interaction

We use the `ConnectionFactory` method `getConnection()` to get our `Connection`. We then use the `Connection` method `createInteraction()` to get an `Interaction` that can be used to flow a request to CICS. If either method call throws an exception, we print the stack trace to the application server and then throw a new exception explaining what happened, with the original exception details.

The next method in our session bean is `getNonManagedConnection()`. This gets a connection to CICS that is not managed by a connection factory installed into the application server. The method instantiates the necessary classes directly and sets some details of the ECI request directly, such as the CICS server name. The first section of the code is shown in Figure B-40 on page 374.

```
try
{
  ECIManagedConnectionFactory mcf = new ECIManagedConnectionFactory();

  // set details
  mcf.setConnectionURL(gatewayURL);
  mcf.setServerName(cicsServer);
  mcf.setPortNumber(Integer.toString(gatewayPort));

  if (username.equals("") == false)
  {
    mcf.setUserName(username);
  }
  if (password.equals("") == false)
  {
    mcf.setPassword(password);
  }
  if (mirror.equals("") == false)
  {
    mcf.setTranName(mirror);
  }
```

Figure B-40 Instantiating an ECIManagedConnectionFactory directly

We use the various setter methods on `ECIManagedConnectionFactory` to set the Gateway daemon URL, Gateway daemon port, CICS server, CICS user ID, CICS password and CICS transaction.

In the next section of code, shown in Figure B-41 on page 375, we set the CICS ECI resource adapter trace level and set trace output to go to standard error. We use the `createConnectionFactory()` method of `ECIManagedConnectionFactory` to create a `ConnectionFactory`, then we get a `Connection` and `Interaction` using the same methods as for a managed connection factory.

```
// set CCI trace
switch (trace)
{
  case 0 :
    mcf.setTraceLevel(new Integer(mcf.RAS_TRACE_OFF));
    break;
  case 1 :
    mcf.setTraceLevel(new Integer(mcf.RAS_TRACE_ERROR_EXCEPTION));
    break;
  case 2 :
    mcf.setTraceLevel(new Integer(mcf.RAS_TRACE_ENTRY_EXIT));
    break;
  case 3 :
    mcf.setTraceLevel(new Integer(mcf.RAS_TRACE_INTERNAL));
    break;
  default :
}

// set trace output to stderr
mcf.setLogWriter(new PrintWriter(System.err));

ConnectionFactory cf = (ConnectionFactory) mcf.createConnectionFactory();
eciConn = cf.getConnection();
eciInt = eciConn.createInteraction();
```

Figure B-41 Setting CCI trace and getting a Connection and Interaction

The final method in our session bean is `dropConnection()`, which closes our connection to CICS. As shown in Figure B-42 on page 376 we call the `close()` method on the `Connection` and `Interaction`, then set the references to `null`.

```
private void dropConnection()
{
  // tidy up the interaction and connection and set our references to
  // null.
  try
  {
    eciInt.close();
    eciConn.close();
  }
  catch (Exception e)
  {
    e.printStackTrace();
  }
  eciInt = null;
  eciConn = null;
}
```

Figure B-42 Closing the connection to CICS

JSPs

The application uses two JSPs to format and display the results. One of two scenarios can occur; the request completes successfully, or an exception occurs inside the CICS ECI resource adapter or our code. The JSPs are stored in the CTGTesterCCIWeb enterprise application project in WebSphere Studio, under the webApplication folder.

results.jsp

The JSP results.jsp processes successful completion of a request. The JSP defines the title of the HTML page and the style sheet used to format the HTML in the header. The JSP defines the JavaBeans components that the page uses to format and display the results (Figure B-43).

```
<jsp:useBean class="java.lang.String" id="results" scope="request"/>
<jsp:useBean class="java.lang.String" id="commareaInput" scope="request"/>
<jsp:useBean class="java.lang.String" id="encoding" scope="request"/>
<jsp:useBean class="java.lang.String" id="defaultResults" scope="request"/>
<jsp:useBean class="java.lang.String" id="hexResults" scope="request"/>
```

Figure B-43 JavaBeans components used by results.jsp

These components are all in the request scope and are all `String` objects. Their IDs correspond to the names of the attributes we set in the `HttpRequest` in our servlet. They contain the result of the CICS request made by the servlet.

The page then includes our common JSP header, which displays the information specified in the initial form, such as trace level and number of iterations. The rest of the page then outputs the values of the request attributes using the scriptlet format `<%= JavaBean id %>`, shown in Figure B-44. In our application, the JavaBean component `encoding` contains the encoding used to convert the input COMMAREA into bytes, and is inserted into the HTML output with the JSP scriplet `<%= encoding %>`.

```
<TABLE BGCOLOR=white CELLPADDING=0><TBODY>
<TR ALIGN=LEFT><TH>Results </TH><TH> COMMAREA </TH></TR>
<TR><TD>Input:</TD><TD BGCOLOR=yellow><%= commareaInput %></TD></TR>
<TR><TD>Output using <%= encoding %>:</TD><TD BGCOLOR=yellow><%= results
%></TD></TR>
<TR><TD>Output using default encoding: </TD><TD BGCOLOR=yellow><%=
defaultResults %></TD></TR>
<TR><TD>Output in HEX: </TD><TD BGCOLOR=yellow><%= hexResults %></TD></TR>
</TBODY></TABLE>
```

Figure B-44 Outputting the results into the HTML

The results page outputs the COMMAREA data converted to a `String` using the JVM default encoding, the encoding specified in the form and in a hexadecimal representation.

error.jsp

The JSP error.jsp is used when an error occurs inside the CICS ECI resource adapter or our code, resulting in an exception being thrown. It does not display the output COMMAREA. Instead, it displays the contents of the exception that occurred and any errors found in the processing of the form parameters by the servlet.

Application XML

The XML that describes the enterprise application can be examined from within WebSphere Studio.

Web deployment descriptor

To view the Web application XML, perform the following steps from WebSphere Studio:

1. From J2EE Perspective Navigator, right-click **CTGTesterCCIWeb** and select **Edit Deployment Descriptor**.

This displays the Web deployment descriptor editor's General tab. Click the **Servlets** tab to show the Servlets pane. Click **CTGTesterCCIServlet** in the Servlets list (Figure B-45 on page 378).

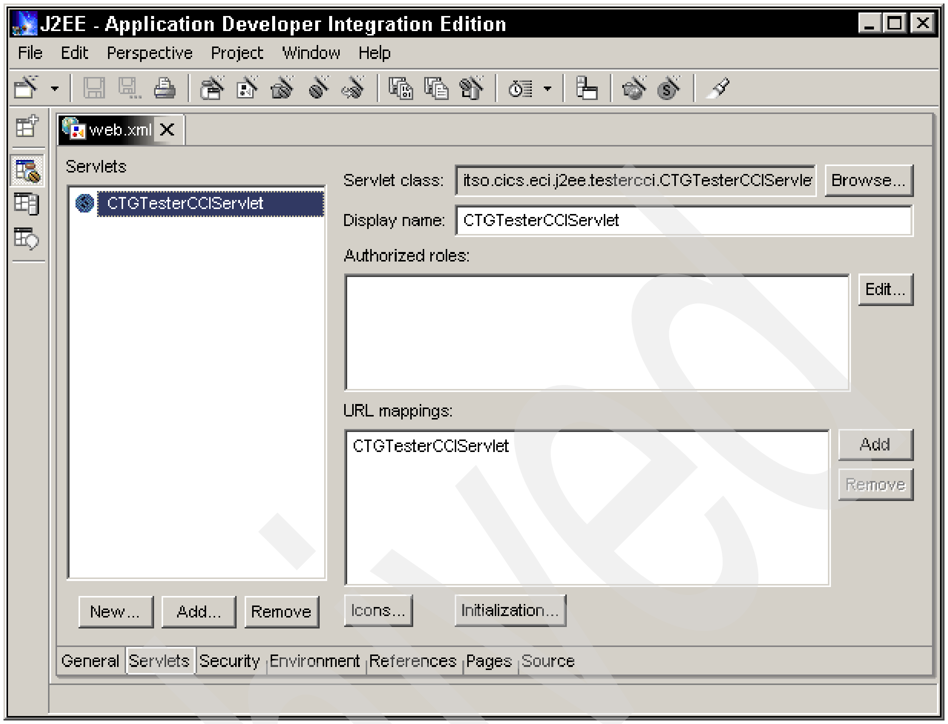


Figure B-45 CTGTesterCCI Web deployment descriptor servlets

We can see that the Servlet class is `itso.cics.eci.j2ee.testercci.CTGTesterCCIServlet`, the class we have outlined in the previous section. The display name is `CTGTesterCCIServlet`. The URL mappings section has one entry; `CTGTesterCCIServlet`. This means that a URL of `CTGTesterCCIServlet` relative to the web application root will map to this servlet. Because of the Servlet class entry, this URL will invoke the class `itso.cics.eci.j2ee.testercci.CTGTesterCCIServlet`.

There is nothing of note on the Security or Environment tabs.

Click **References** to show the References pane, as shown in Figure B-46 on page 379.

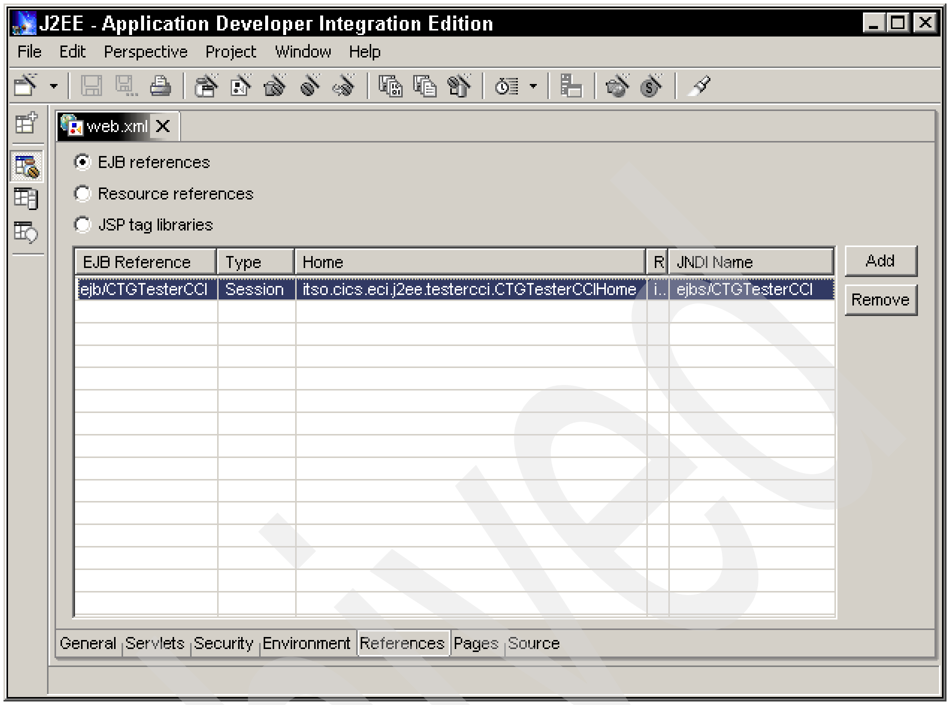


Figure B-46 CTGTesterCCI Web deployment descriptor references

As shown, there is an EJB reference defined in the Web deployment descriptor. The EJB Reference is `ejb/CTGTesterCCI`, the JNDI name we used in our servlet to look up our session bean. The expected Java class type of the referenced bean is `Session`, because our bean is a session bean. The Home field contains the fully qualified name of the enterprise bean's home interface, which is `itso.cics.eci.j2ee.testercci.CTGTesterCCIHome`. This class is defined in the CTGTesterCCIEJB project. The Remote field contains the fully qualified name of the enterprise bean's remote interface, which is `itso.cics.eci.j2ee.testercci.CTGTesterCCI`. Again this class is defined in the CTGTesterCCIEJB project. The JNDI Name field contains the JNDI name that the enterprise bean is expected to be bound with during runtime. We have defined a suggested name of `ejbs/CTGTesterCCI` here, but this can be changed at deploy time. In the case of WebSphere for z/OS, it is necessary to change this reference at deploy time because the JNDI name of our session bean will not be `ejbs/CTGTesterCCI`.

Click the **Pages** tab to show the Pages pane. We have an entry `index.jsp` in the Welcome files pane. This means that a URL that specifies the Web application root, for example "/", will cause the page index.jsp to be looked for in the Web application. We supply an index.jsp, so this page will be found and displayed in the Web browser.

Click the **Source** tab to display the Source panel. This shows the source of the `web.xml` file that is generated from entries in the Web deployment descriptor editor (Figure B-47).

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app id="WebApp">
   <display-name>CTGTesterCCIWeb</display-name>
   <servlet>
      <servlet-name>CTGTesterCCIServlet</servlet-name>
      <display-name>CTGTesterCCIServlet</display-name>

<servlet-class>itso.cics.eci.j2ee.testercci.CTGTesterCCIServlet</servlet-cla
ss>
   </servlet>
   <servlet-mapping>
      <servlet-name>CTGTesterCCIServlet</servlet-name>
      <url-pattern>CTGTesterCCIServlet</url-pattern>
   </servlet-mapping>
   <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
   </welcome-file-list>
   <ejb-ref id="EjbRef_1">
      <description></description>
      <ejb-ref-name>ejb/CTGTesterCCI</ejb-ref-name>
      <ejb-ref-type>Session</ejb-ref-type>
      <home>itso.cics.eci.j2ee.testercci.CTGTesterCCIHome</home>
      <remote>itso.cics.eci.j2ee.testercci.CTGTesterCCI</remote>
   </ejb-ref>
</web-app>
```

Figure B-47 CTGTesterCCI Web deployment descriptor source

As can be seen, the display name of our Web application is specified with the `<display-name>` tag. Most of the values shown in the Web deployment descriptor editor are marked up using the same name, except for the servlet mapping, which is marked up with the `<servlet-name>` tag to show which servlet the `<url-pattern>` corresponds to. The EJB reference is defined using the tag `<ejb-ref>` with an `id` attribute generated by the editor, in this case `EjbRef_1`.

All the fields we defined in the editor are here except for the JNDI Name field. This is stored in the Web app binding XMI file. To view this file, from the J2EE Perspective Navigator view, expand **CTGTesterCCIWeb -> webApplication ->**

WEB-INF, right-click **ibm-web-bnd.xmi**, and select **Open**. Click the **Source** tab to see the source, shown in Figure B-48.

```
<?xml version="1.0" encoding="UTF-8"?>
<webappbnd:WebAppBinding xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns:webappbnd="webappbnd.xmi"
xmlns:webapplication="webapplicaiton.xmi" xmlns:commonbnd="commonbnd.xmi"
xmlns:common="common.xmi" xmi:id="WebAppBinding_1"
virtualHostName="default_host">
  <webapp href="WEB-INF/web.xml#WebApp"/>
  <ejbRefBindings xmi:id="EjbRefBinding_1" jndiName="ejbs/CTGTesterCCI">
    <bindingEjbRef href="WEB-INF/web.xml#EjbRef_1"/>
  </ejbRefBindings>
</webappbnd:WebAppBinding>
```

Figure B-48 ibm-web-bnd.xmi source

As shown, the tag <ejbRefBindings> has an attribute that defines the JNDI name to use for our EJB reference. The attribute is jndiName and its value is ejbs/CTGTesterCCI, as we specified in the Web deployment descriptor editor. The <bindingEjbRef> tag refers to the EJB reference EjbRef_1, as defined by the editor in web.xml. This ties the JNDI name defined in the ibm-web-bnd.xmi file to the EJB reference in the web.xml file.

Web context root

The context root of our Web application can be viewed and edited from the Web pane in the enterprise application properties window in WebSphere Studio. To see this pane, perform the following steps:

1. From the J2EE Perspective Navigator view, right-click **CTGTesterCCIWeb** and select **Properties**.
2. From the Properties dialog tree, click the **Web** node. This will show the Web pane (Figure B-49 on page 382).

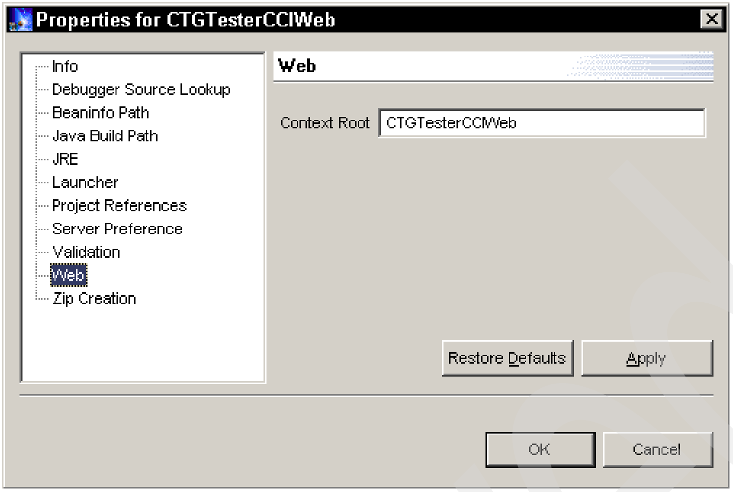


Figure B-49 CTGTesterCCIWeb properties Web pane

As shown in the Context Root field, the context root for our Web application is `CTGTesterCCIWeb`. Therefore the URL to invoke the application is `/CTGTesterCCIWeb`, assuming that, at deploy time, the Application Server is set to map enterprise applications directly to /. Because the servlet URL mapping is relative to the context root, the URL `/CTGTesterCCIWeb/CTGTesterCCIServlet` can be used to directly invoke the servlet.

EJB deployment descriptor

To view the EJB deployment descriptor, perform the following steps from WebSphere Studio:

1. From the J2EE Perspective Navigator view, expand **CTGTesterCCIEJB -> ejbModules -> META-INF**.
2. Right-click **ejb-jar.xml** and select **Open**.

The General tab shows some information about the session bean. Click the **Beans** tab to show the Beans pane. This shows we have one bean defined, `CTGTesterCCI`, and that it is a Stateless Session bean. The Bean class is `CTGTesterCCIBean`, the bean implementation that we outlined in "Session bean" on page 367. The Home interface is `CTGTesterCCIHome` and the Remote interface is `CTGTesterCCI`.

Click the **Transaction** to show the Transaction pane. Expand **CTGTesterCCI** (Figure B-50 on page 383).

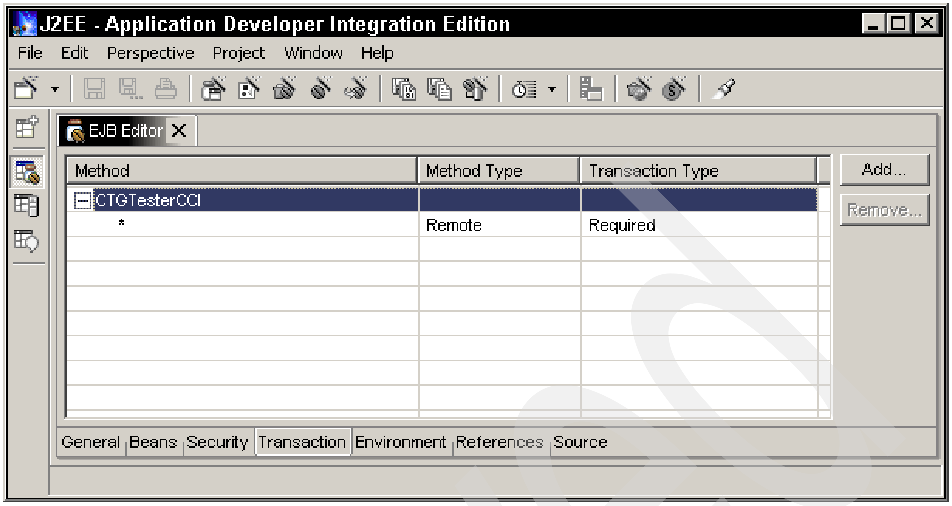


Figure B-50   CTGTesterCCI bean transaction properties

As shown, all remote methods on the bean have a Transaction Type of `Required`. This means transactionality is required when these methods execute.

Click the **References** tab to show the References pane. To see our connection factory reference, select the **Resource references** radio button and then expand **CTGTesterCCI**, as shown in Figure B-51.

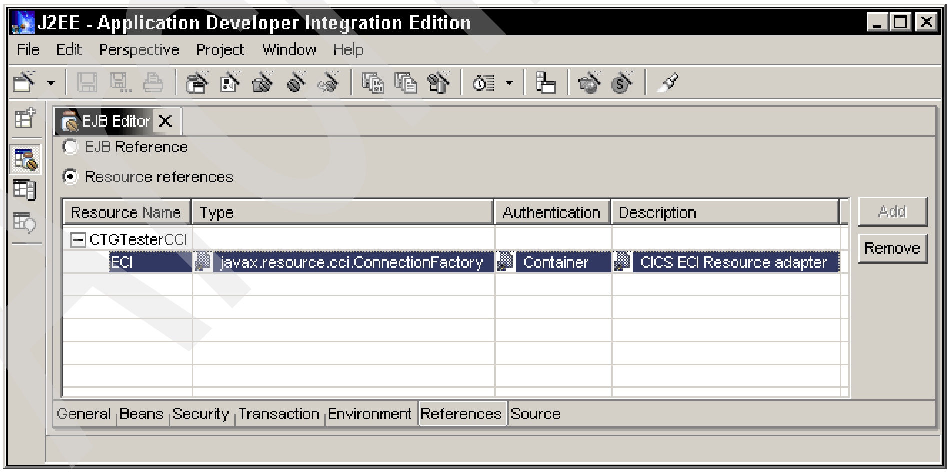


Figure B-51   CTGTesterCCI bean resource references

The Resource Name is `ECI`. This is the local JNDI name used in our session bean to look up the CICS connection factory, as outlined in “Session bean” on page 367. The Type is javax.resource.cci.ConnectionFactory, a CCI connection factory. The JNDI name that this is mapped to is not defined in the EJB deployment descriptor, rather in the extension file ibm-ejb-jar-bnd.xmi.

Click the **Source** tab to show the Source pane. The source of the EJB deployment descriptor is shown in this pane and in Figure B-52.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar id="ejb-jar_ID">
   <display-name>CTGTesterCCIEJB</display-name>
   <enterprise-beans>
      <session id="CTGTesterCCI">
         <ejb-name>CTGTesterCCI</ejb-name>
         <home>itso.cics.eci.j2ee.testercci.CTGTesterCCIHome</home>
         <remote>itso.cics.eci.j2ee.testercci.CTGTesterCCI</remote>

<ejb-class>itso.cics.eci.j2ee.testercci.CTGTesterCCIBean</ejb-class>
         <session-type>Stateless</session-type>
         <transaction-type>Container</transaction-type>
         <resource-ref id="ResourceRef_1">
            <description>CICS ECI Resource adapter</description>
            <res-ref-name>ECI</res-ref-name>
            <res-type>javax.resource.cci.ConnectionFactory</res-type>
            <res-auth>Container</res-auth>
         </resource-ref>
      </session>
   </enterprise-beans>
   <assembly-descriptor>
      <container-transaction>
         <method>
            <ejb-name>CTGTesterCCI</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>*</method-name>
         </method>
         <trans-attribute>Required</trans-attribute>
      </container-transaction>
   </assembly-descriptor>
</ejb-jar>
```

Figure B-52   CTGTesterCCI EJB deployment descriptor source

As shown, most of the names used in the EJB Editor are the same as the tags in the source. The resource reference is defined with the tag <resource-ref>. Its id is ResourceRef_1.

To show what the resource reference is bound to, it is necessary to open the ibm-ejb-jar-bnd.xmi file. To view this file, from the J2EE Perspective Navigator view, expand **CTGTesterCCIEJB -> ejbModule -> META-INF**, right-click **ibm-ejb-jar-bnd.xmi**, and select **Open**. Click the **Bindings** tab to see the Bindings pane.

To see the resource reference binding, from the Bindings pane expand **CTGTesterCCIEJB -> CTGTesterCCI -> ResourceRef ECI** and click **ResourceRef ECI**. As shown in Figure B-53, the resource reference `ECI` is bound to the JNDI name `eis/CICSA`. This can be changed to a deployed connection factory at deploy time. However, when testing we had to define a connection factory in the test server with the JNDI name `eis/CICSA` so that this reference would be valid and we could test using managed CICS connections.

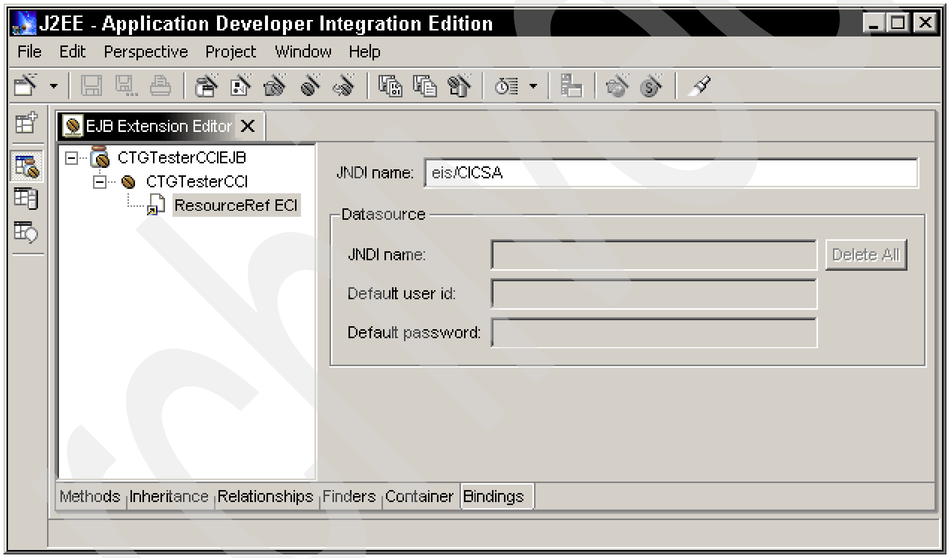


Figure B-53 CTGTesterCCI bean resource reference binding

To see the JNDI name our session bean is bound to, from the Bindings pane click **CTGTesterCCI**. Figure B-54 on page 386 shows that our bean has a suggested JNDI name of `ejbs/CTGTesterCCI`. This is the same name we used in our Web deployment descriptor as the JNDI name of the session bean, in Figure B-46 on page 379. Because these names are the same, the servlet will look up our session bean using the same JNDI name as it was bound with. Therefore the servlet will find our session bean.

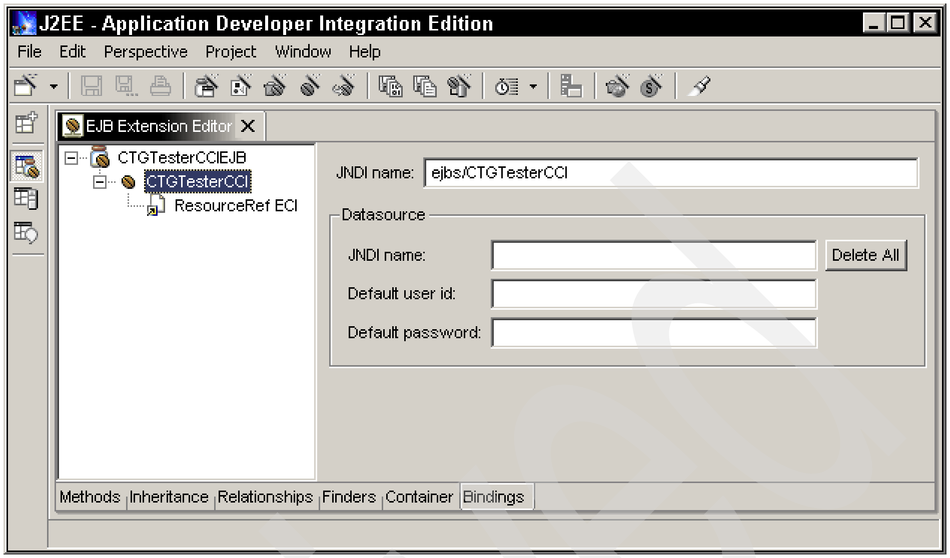


Figure B-54 CTGTesterCCI bean JNDI name

The source of the ibm-ejb-jar-bnd.xmi file can be viewed from WebSphere Studio by right-clicking **ibm-ejb-jar-bnd.xmi** and selecting **Open With -> XML Editor**. The source is shown in Figure B-55.

```
<?xml version="1.0" encoding="UTF-8"?>
<ejbbnd:EJBJarBinding xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:ejbbnd="ejbbnd.xmi" xmlns:ejb="ejb.xmi"
xmlns:commonbnd="commonbnd.xmi" xmlns:common="common.xmi"
xmi:id="ejb-jar_ID_Bnd">
  <ejbJar href="META-INF/ejb-jar.xml#ejb-jar_ID"/>
  <ejbBindings xmi:id="CTGTesterCCI_Bnd" jndiName="ejbs/CTGTesterCCI">
    <enterpriseBean xmi:type="ejb:Session"
href="META-INF/ejb-jar.xml#CTGTesterCCI"/>
    <resRefBindings xmi:id="ResourceRefBinding_1" jndiName="eis/CICSA">
      <bindingResourceRef href="META-INF/ejb-jar.xml#ResourceRef_1"/>
    </resRefBindings>
  </ejbBindings>
</ejbbnd:EJBJarBinding>
```

Figure B-55 Source of ibm-ejb-jar-bnd.xmi

The tag `<resRefBindings>` denotes the binding for our resource reference. The attribute `jndiName` is `eis/CICSA`, corresponding to what we specified in the editor. The tag `<bindingResourceRef>` shows which resource reference this binding specifies. The `href` attribute points to the resource reference `ResourceRef_1` in the EJB deployment descriptor.

Application deployment descriptor

To view the application deployment descriptor, perform the following steps from WebSphere Studio:

1. From the J2EE Perspective Navigator view, expand **CTGTesterCCI -> META-INF**.
2. Right-click **application.xml** and select **Open**.

The General tab shows some information about the application. Click the **Modules** tab to show the Modules pane. The Modules panel defines which modules make up the enterprise application and shows which URL invokes the application. Click **CTGTesterCCIWeb.war** (Figure B-56).

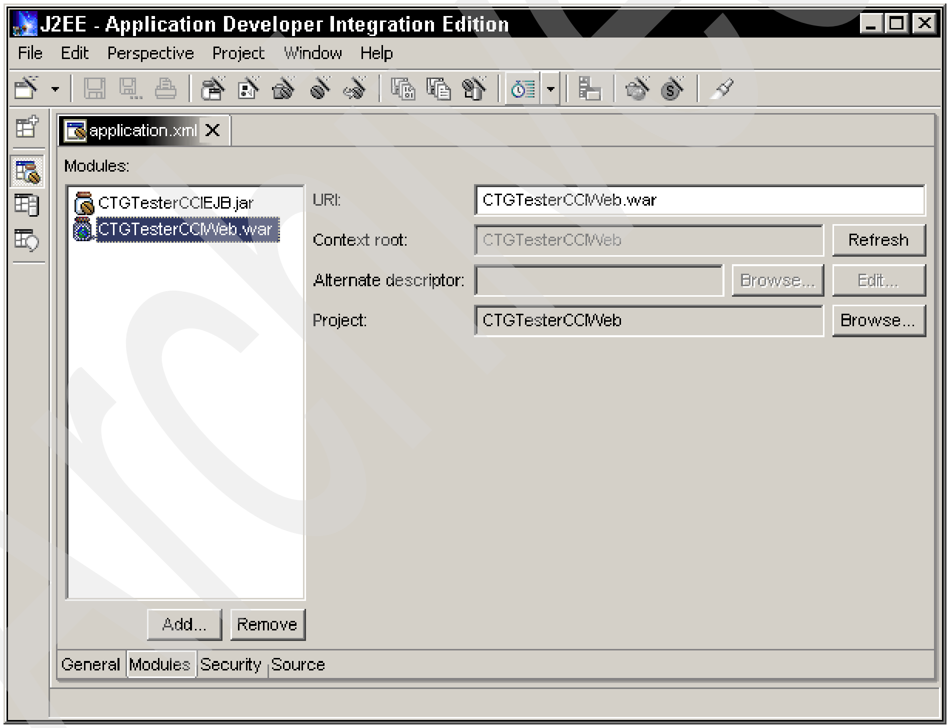


Figure B-56 CTGTesterCCI application.xml modules panel

As shown, the application uses the CTGTesterCCIWeb module. This is a Web application, so it is contained in a Web Archive Resource (WAR) file. When CTGTesterCCIWeb is built, WebSphere Studio builds it into a WAR file with the name of the Web application, and makes it available to the CTGTesterCCI application. The URI of this WAR file is therefore CTGTesterCCIWeb.war. The Context root field is set to CTGTesterCCIWeb, from the CTGTesterCCIWeb application properties. The EJB module is contained in CTGTesterCCIEJB.jar.

C

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG246133`

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246133, and select the zip file for the second edition of this book, SG246133-01src.zip.

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	2 MB minimum
Operating System:	Windows NT or 2000
Processor:	Intel Pentium
Memory:	64 MB

Using the Web material

The additional Web material (`SG246133-01src.zip`) that accompanies this redbook contains the following directory structure:

- **CICS** directory containing the following CICS source code:
  - EPIPROG.TXT (as used in Chapter 6)
  - EPIMAPS.BMS (as used in Chapter 6)
  - ECIPROG2.TXT (as used in Chapter 10 and Chapter 11)
- **EJB_components** directory containing the following EAR files for our sample Web applications as used in Chapter 10 and Chapter 11:
  - CTGTesterECI.ear
  - CTGTesterCCI.ear

  For further details on these applications, refer to Appendix B, “Sample applications” on page 337.
- **Java** directory containing the Java source code tree for the following packages:
  - `itso.cics.eci.j2ee.testercci`
  - `itso.cics.eci.testereci`
  - `itso.cics.epi`

  The source for the `itso.cics.eci.j2ee.testercci` and `itso.cics.eci.testereci` packages are also supplied in the EAR files in the EJB_components directory. The package `itso.cics.epi` contains the SignonCapable.java and SignonIncapable.java applications used in conjunction with the CICS program EPIPROG in Chapter 6.
- **Utilities** directory containing the following files:

JNDIView.zip	JNDI viewer zip file
listJNDI.cmd	DOS cmd file to start JNDIview
tcp62locallu.exe	TCP62 LU generation utility

  For details on using tcp62locallu, refer to Chapter 3. For details on using JNDIView refer to the supplied readme.jndivew.txt file.

If you wish to follow the examples in this redbook, create a subdirectory called itsoctgv5 on your workstation, and unzip the contents of the Web material zip file into this folder.

Importing CTGTesterECI into WebSphere Studio

WebSphere Studio Application Developer Integration Edition (WebSphere Studio) includes the CICS TG resource adapters and an integrated WebSphere test environment. It is possible to edit, compile and run both our sample enterprise applications from within WebSphere Studio.

To import the CTGTesterECI EAR file into WebSphere Studio Application Developer Integration Edition perform the following steps:

1. Select **File -> Import**. Select **EAR file** from the Select an import source window. Click **Next**.
2. Click the **Browse** button next to the EAR File field. In the Open window, navigate to the file CTGTesterECI.ear, select it and click **Open**.
3. In the field Enterprise Application project name, enter `CTGTesterECI` (as shown in Figure C-1). Click **Finish**.

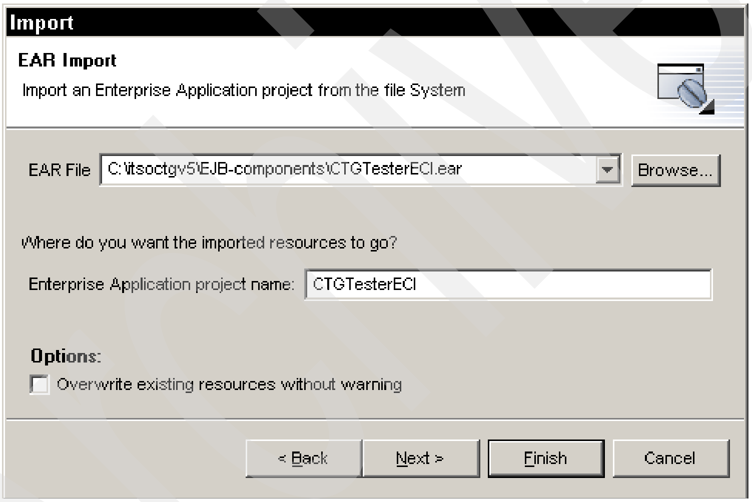


Figure C-1 WebSphere Studio Import window

WebSphere Studio will import the contents of the EAR into two enterprise application projects: CTGTesterECI and CTGTesterECIWeb. CTGTesterECI is the enterprise project. CTGTesterECIWeb contains the Web deployment descriptor, JSPs and servlet source code. It will also contain the compiled Java code that was present in the EAR file. This is not really needed, because the application can be compiled from the source code. Perform the following steps to remove the compiled Java code:

1. From the J2EE perspective Navigator, click **CTGTesterECIWeb -> webApplication -> WEB-INF -> lib**.

2. Right-click **CTGTesterECIWeb_classes.jar** and select **Delete** from the pop-up menu.
3. Click **Yes** on the Delete Resources window.

In order to compile the application, the CICS TG Java class library ctgclient.jar has to be added to the project. This can be added from the CICS ECI resource adapter in WebSphere Studio. To add the CICS TG Java class library, perform the following steps:

1. Right-click the **CTGTesterECIWeb** enterprise application folder and select **Properties**.
2. From the Properties window, click the **Java Build Path** node.
3. Click the **Libraries** tab. This will show the Libraries window (see Figure C-2).

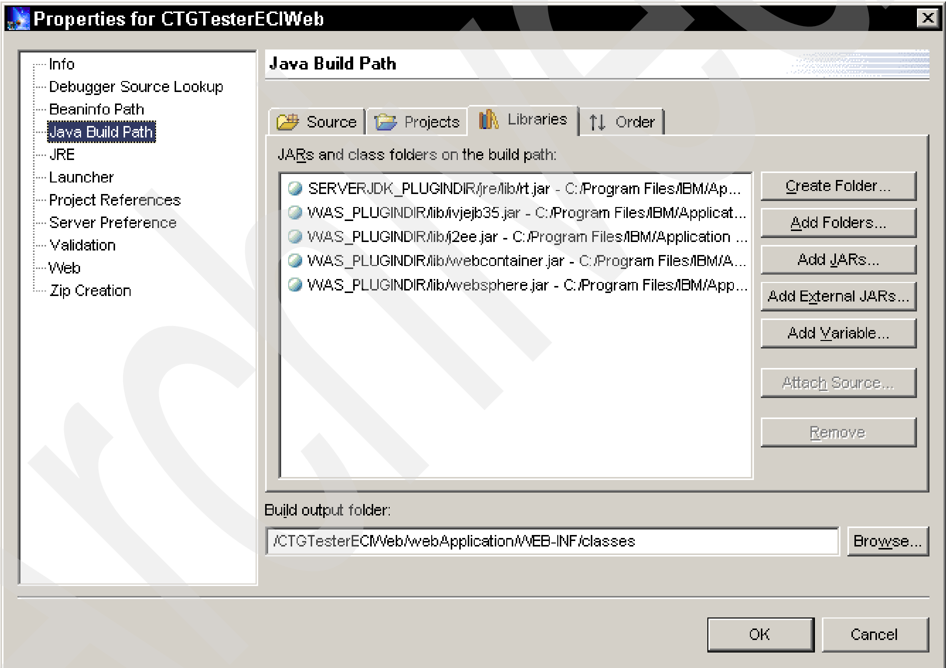


Figure C-2 WebSphere Studio Java Build Path window

4. Click **Add JARs**.
5. In the JAR Selection window, expand **Installed Resource Adapters -> CICS ECI**.
6. Select **ctgclient.jar** and click **OK**.
7. From the Properties window, click **OK**.

The CTGTesterECIWeb application should now compile without warnings.

Testing

To test CTGTesterECI using WebSphere Studio, right-click the **CTGTesterECIWeb** enterprise application folder, and select **Run On Server**.

If there are no servers defined, this will create a test server configuration called WebSphere Administrative Domain and a test instance called WebSphere V4.0 Test Environment.

The Publishing window will appear, followed by the Server window. The Console view will show the integrated WebSphere server starting. Once the server is started (shown by the message `Server Default Server open for e-business`) the Web browser view will open at the CTGTesterECI welcome page (Figure C-3).

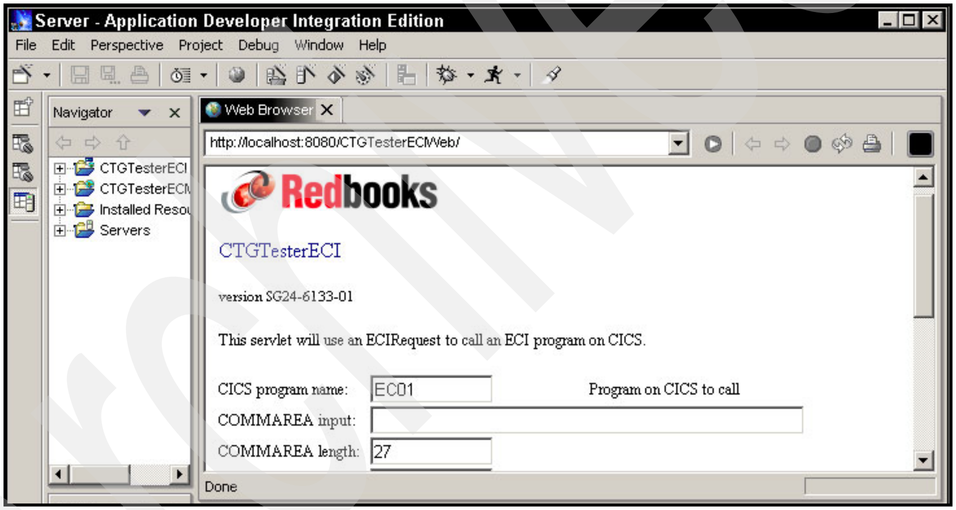


Figure C-3 WebSphere Studio testing CTGTesterECI

In order to flow a request to CICS, the server instance must have the CICS TG Java class libraries added. The easiest way to do this is to add the CICS ECI resource adapter to the WebSphere server instance. To do this, perform the following steps:

1. From the Server perspective Server Configuration view, expand **Server Configurations**.
2. Right-click **WebSphere Administrative Domain** and select **Open**.
3. In the WebSphere Administrative Domain window that appears, click the **J2C** tab.

4. In the J2C Resource Adapters section, click **Add**. The Create Resource Adapter window will appear (Figure C-4).

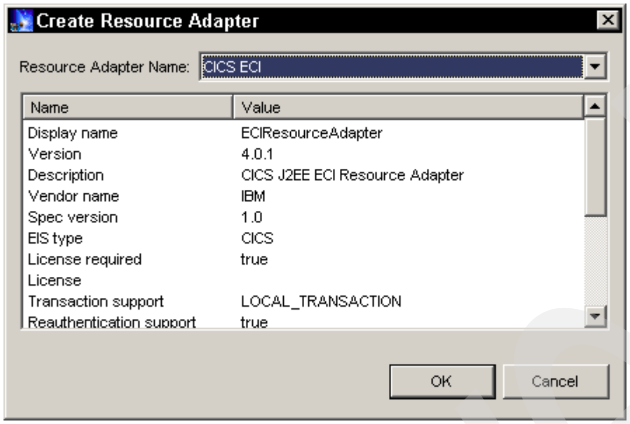


Figure C-4 Create Resource Adapter window

5. In the Resource Adapter Name drop-down, select **CICS ECI**.
6. Click **OK**. The WebSphere Administrative Domain window will look like Figure C-5 on page 395.

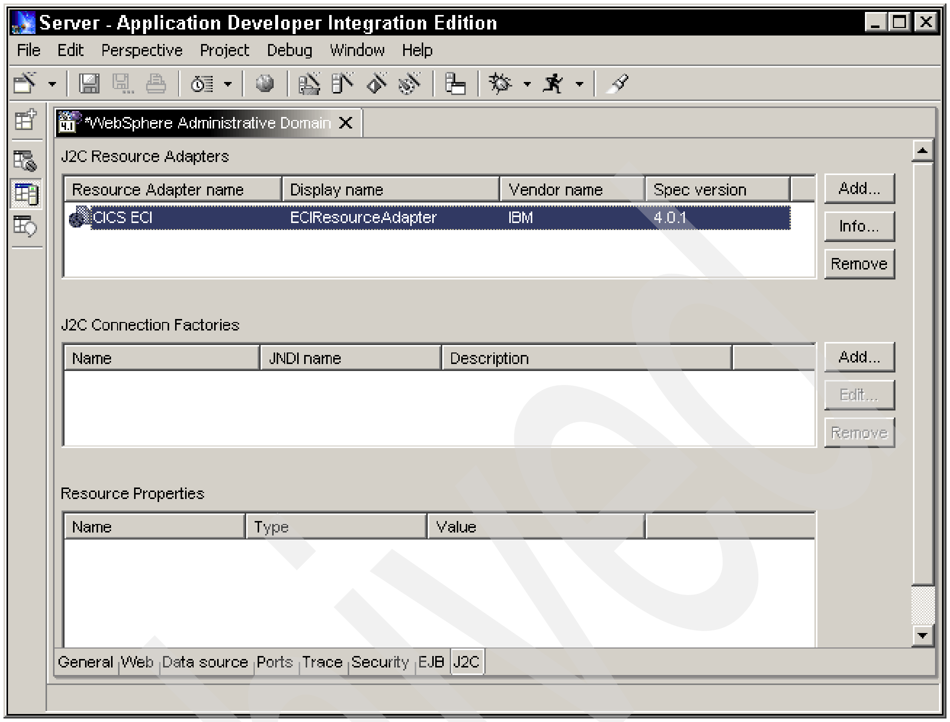


Figure C-5 WebSphere Studio server configuration window

7. Select **File -> Save WebSphere Administrative Domain** to save the changes.

The test server must be restarted and republished for these changes to take effect. To do this, perform the following steps:

1. From the Server perspective Servers view, click the **Servers** tab.
2. Right-click the server instance **WebSphere V4.0 Test Environment**. Click **Stop**.
3. Once the server has stopped, right-click the server instance again and select **Publish**. The Publishing window will appear.
4. Once publishing has completed, from the Publishing window click **OK**.
5. Right-click the server instance again and select **Start**. The Server window will appear.

The application can now be tested. For details on how to deploy and test CTGTesterECI with WebSphere for Windows, refer to Chapter 11. For details on how to deploy and test CTGTesterECI with WebSphere for z/OS, refer to Chapter 10.

Importing CTGTesterCCI into WebSphere Studio

To import the CTGTesterCCI EAR file into WebSphere Studio Application Developer Integration Edition, perform the following steps:

1. Select **File -> Import**. Select **EAR file** from the Select an import source window. Click **Next**.
2. Click the **Browse** button next to the EAR File field. In the Open window, navigate to the file CTGTesterCCI.ear, select it and click **Open**.
3. In the field Enterprise Application project name, enter `CTGTesterCCI` (Figure C-6). Click **Finish**.

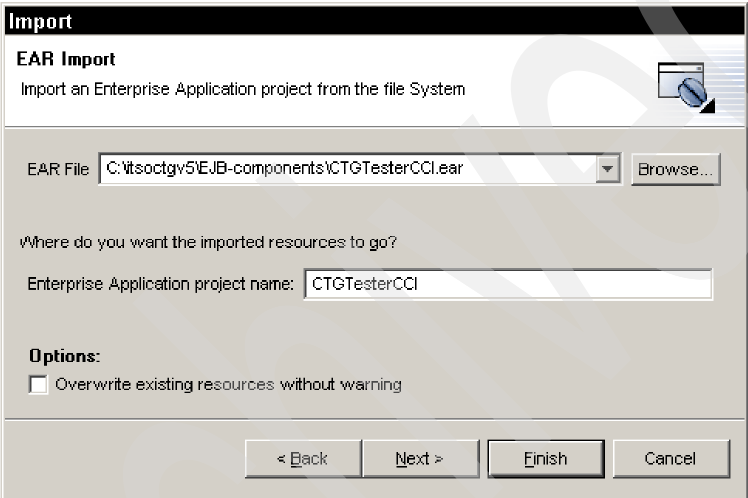


Figure C-6 WebSphere Studio Import window

WebSphere Studio will import the contents of the EAR into three enterprise application projects: CTGTesterCCI, CTGTesterCCIEJB, and CTGTesterCCIWeb. CTGTesterCCI is the enterprise project that references the associated Web and EJB projects. CTGTesterCCIEJB contains the EJB deployment descriptor and session bean source code. CTGTesterCCIWeb contains the Web deployment descriptor, JSPs, and servlet source code. The compiled Java code that was present in the EAR file is also imported. This is not really needed, because the application can be compiled from the source code. Perform the following steps to remove the compiled Java code:

1. From the J2EE perspective Navigator, expand **CTGTesterCCIWeb -> webApplication -> WEB-INF -> lib**.
2. Right-click **CTGTesterCCIWeb_classes.jar** and select **Delete** from the pop-up menu.
3. Click **Yes** on the Delete Resources window.

4. Expand **CTGTesterCCIEJB -> ejbModule**.
5. Right-click **CTGTesterCCIEJB.imported_classes.jar** and select **Delete** from the pop-up menu.
6. Click **Yes** on the Delete Resources window.
7. Right-click the **CTGTesterCCIEJB** enterprise application folder and select **Edit Module Dependencies**. The Module Dependencies window will appear with the imported_classes.jar selected as a dependency (Figure C-7).

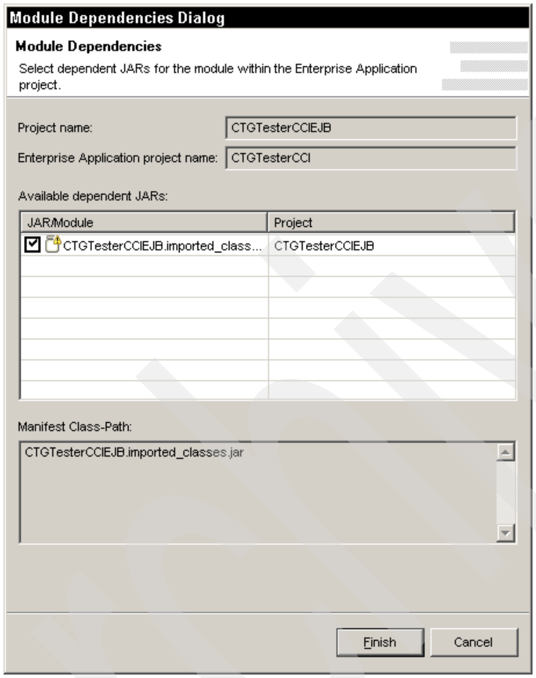


Figure C-7 Edit module dependencies window

8. Uncheck **CTGTesterCCI.imported_classes.jar**.
9. Click **Finish**.

In order to compile the application, the CICS ECI resource adapter Java class libraries have to be added to the EJB project. These can be added from the CICS ECI resource adapter in WebSphere Studio. The J2EE Connector Architecture library also has to be added to the EJB and Web projects. To add the required Java classes to the EJB project, perform the following steps:

1. Right-click the **CTGTesterCCIEJB** enterprise application folder and select **Properties**.
2. From the Properties window, click the **Java Build Path** node.

3. Click the **Libraries** tab. This will show the Libraries window (see Figure C-2 on page 392).

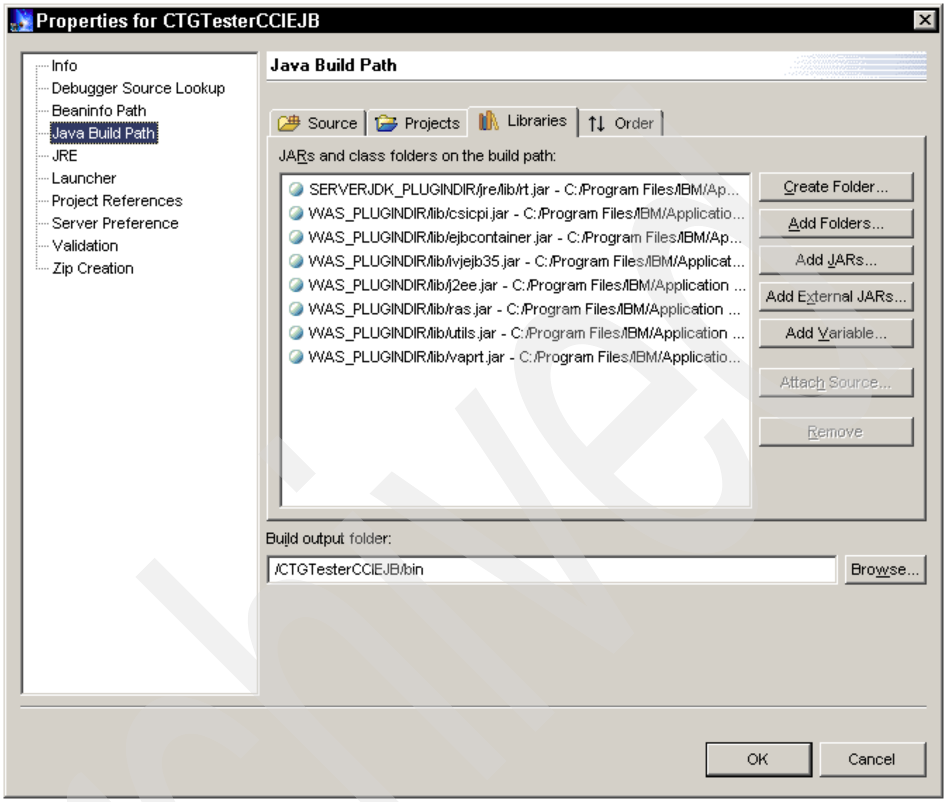


Figure C-8 WebSphere Studio Java Build Path window

4. Click **Add JARs**.
5. In the JAR Selection window, expand **Installed Resource Adapters -> CICS ECI**.
6. Select **ctgclient.jar**, **cicseci.jar** and **cicsframe.jar** and click **OK**. Multiple JAR files can be selected by holding down the Ctrl key when selecting.
7. Click **Add Variable**. The Classpath Variable Selection window will appear (Figure C-9 on page 399).

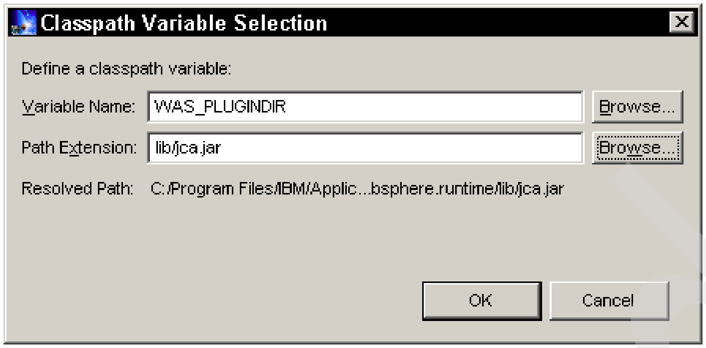


Figure C-9 WebSphere Studio Classpath Variable Selection window

8. Click the **Browse** button next to the Variable Name field. The Variable Selection window will appear.
9. From the Variable Selection window, select **WAS_PLUGINDIR**. Click **OK**.
10. From the Classpath Variable Selection window, click the **Browse** button next to the Path Extension field. The JAR Selection window will appear.
11. From the JAR Selection window, select the **lib** directory and click **Open**.
12. Select **jca.jar** and click **Open**.
13. From the Classpath Variable Selection window, click **OK**.
14. From the Properties window, click **OK**.

The CTGTesterCCIEJB application should now compile without warnings. To add the required Java classes to the Web project, perform the following steps:

1. Right-click the **CTGTesterCCIWeb** enterprise application folder and select **Properties**.
2. From the Properties window, click the **Java Build Path** node.
3. Click the **Libraries** tab. This will show the Libraries window.
4. Click **Add Variable**. The Classpath Variable Selection window will appear.
5. Click the **Browse** button next to the Variable Name field. The Variable Selection window will appear.
6. From the Variable Selection window, select **WAS_PLUGINDIR**. Click **OK**.
7. From the Classpath Variable Selection window, click the **Browse** button next to the Path Extension field. The JAR Selection window will appear.
8. From the JAR Selection window, select the **lib** directory and click **Open**.
9. Select **jca.jar** and click **Open**.

10.From the Classpath Variable Selection window, click **OK**.

11.From the Properties window, click **OK**.

The CTGTesterCCIWeb application should now compile without warnings.

Testing

To test CTGTesterCCI using WebSphere Studio, right-click the **CTGTesterCCIWeb** enterprise application folder and select **Run On Server**.

If there are no servers defined, this will create a test server configuration called WebSphere Administrative Domain and a test instance called WebSphere V4.0 Test Environment.

The Publishing window will appear, followed by the Server window. The Console view will show the integrated WebSphere server starting. Once the server is started (shown by the message `Server Default Server open for e-business`) the Web browser view will open at the CTGTesterCCI welcome page (see Figure C-10).

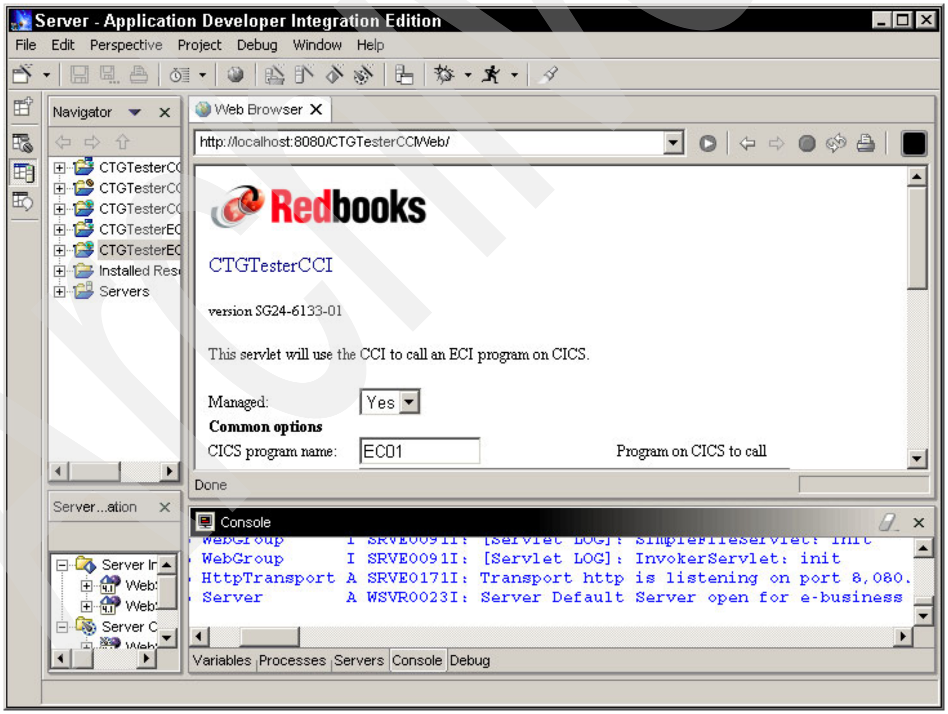


Figure C-10 WebSphere Studio testing CTGTesterCCI

In order to flow a request to CICS, the server instance must have the CICS ECI resource adapter added and a connection factory defined. To do this, perform the following steps:

1. From the Server perspective Server Configuration view, expand **Server Configurations**.
2. Right-click **WebSphere Administrative Domain** and select **Open**.
3. In the WebSphere Administrative Domain window that appears, click the **J2C** tab.
4. In the J2C Resource Adapters section, click **Add**. The Create Resource Adapter window will appear (Figure C-11).

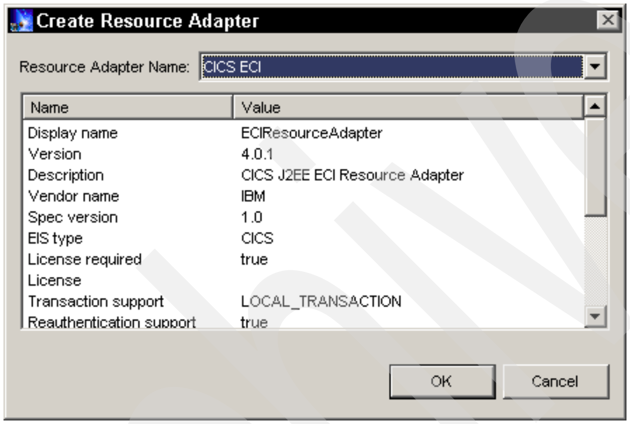


Figure C-11   Create Resource Adapter window

5. In the Resource Adapter Name drop-down, select **CICS ECI** and click **OK**.
6. In the J2C Connection Factories section, click **Add**. The Create Connection Factory window will appear (Figure C-12 on page 402).

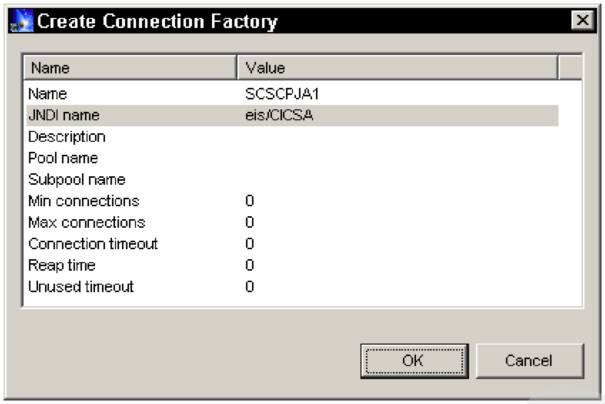


Figure C-12 Create Connection Factory window

7. In the Name field, enter a descriptive name. We used `SCSCPJA1`.
8. In the JNDI name field, enter `eis/CICSA`. Note that this is the JNDI name used by the session bean to look up the connection factory.
9. Click **OK**.
10. In the J2C Connection Factories section, select **SCSCPJA1**. The Resource Properties section will show the connection factory properties.
11. In the Resource Properties section, enter values for a CICS TG and CICS server connection. We used the following values:

    ServerName SCSCPJA1

    ConnectionURL local:
12. The WebSphere Administrative Domain window will now look like Figure C-13 on page 403.

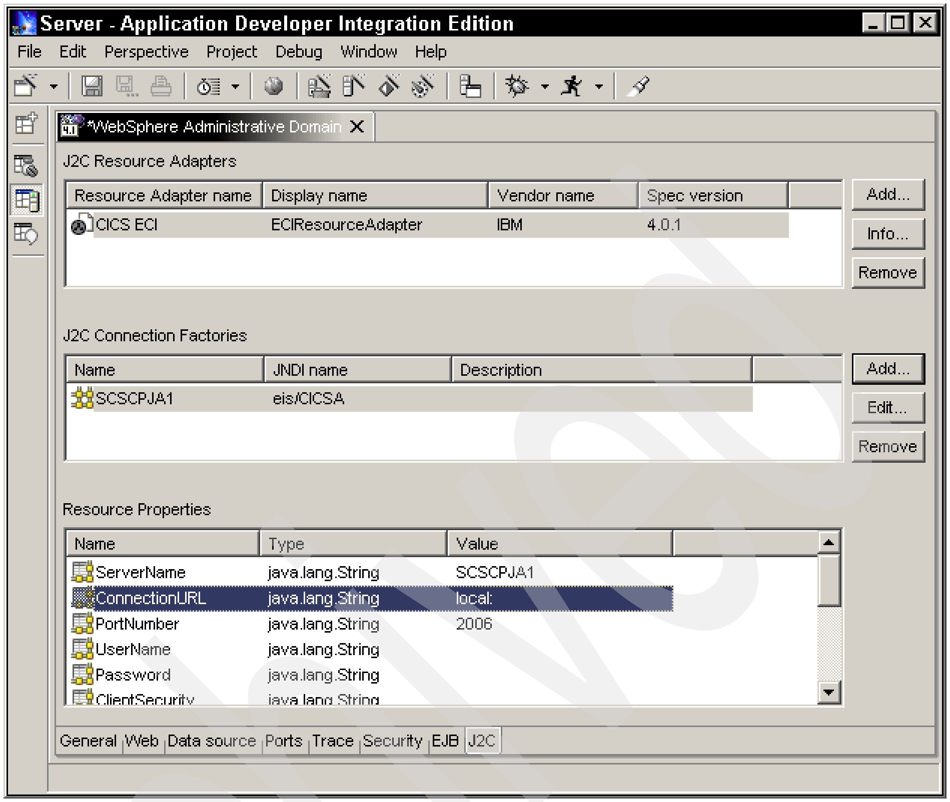


Figure C-13 Server configuration window

13. Select **File -> Save WebSphere Administrative Domain** to save the changes.

The test server must be restarted and republished for these changes to take effect. To do this, perform the following steps:

1. From the Server perspective Servers view, click the **Servers** tab.
2. Right-click the server instance **WebSphere V4.0 Test Environment**. Select **Stop**.
3. Once the server has stopped, right-click the server instance again and select **Publish**. The Publishing window will appear.
4. Once publishing has completed, from the Publishing window click **OK**.
5. Right-click the server instance again and select **Start**. The Server window will appear.

The application can now be tested using managed connections. For details on how to deploy and test CTGTesterCCI with WebSphere for Windows, refer to Chapter 11, “CICS TG and WebSphere Application Server for Windows” on page 289. For details on how to deploy and test CTGTesterCCI with WebSphere for z/OS, refer to Chapter 10, “CICS TG and WebSphere Application Server for z/OS” on page 237.

Abbreviations and acronyms

AOR	application-owning region
API	application programming interface
APPC	advanced program-to-program communications
APPN	Advanced Peer-to-Peer Networking
ASCII	American Standard Code for Information Interchange
AWT	abstract windowing toolkit
BMP	bean-managed persistence
CCF	Common Connector Framework
CCI	Common Client Interface
CMP	container managed persistence
CORBA	Component Object Request Broker Architecture
CICS TG	CICS Transaction Gateway
CSR	certificate signing request
DBMS	database management system
DNS	Domain Name System
DPL	distributed program link
EAB	Enterprise Access Builder
EAR	Enterprise Application Archive
EBCDIC	Extended Binary Coded Decimal Interchange Code
ECI	External Call Interface
EIS	Enterprise Information Systems
EJB	Enterprise JavaBeans
EJS	Enterprise Java Server
EPI	External Presentation Interface
ESI	External Security Interface
ESM	external security manager
EXCI	External CICS Interface
FOR	file-owning region
FTP	File Transfer Protocol
GTF	Generalized Trace Facility
GUI	graphical user interface
HFS	Hierarchical File System
HTML	Hypertext Transfer Protocol
HTTP	Hypertext Markup Language
IRC	inter-region communication
IDE	Integrated development environment
ISC	inter-system communication
J2EE	Java 2 Enterprise Edition
JAR	Java archive
JDBC	Java Database Connectivity
JDK	Java Developer's Kit
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
JSDK	Java Servlet Development Kit
JSP	JavaServer Page
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
LEN	Low Entry Node
LPAR	logical partition
LUW	logical unit of work
MPTN	Multi-Protocol Transport Network
OS/390	operating system 390
RRS	resource recovery services
PEM	password expiration management
SAF	System Authorization Facility
SNA	Systems Network Architecture

Related publications

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

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 408.

- *Java Connectors for CICS*, SG24-6401
- *Revealed! Architecting Web Access to CICS*, SG24-5466
- *Enterprise JavaBeans for z/OS and OS/390, CICS Transaction Server V2.2*, SG24-6284

Other resources

These publications are also relevant as further information sources:

- CICS Transaction Gateway library
  `http://www-3.ibm.com/software/ts/cics/library/manuals/index40.html`
- WebSphere Application Server InfoCenter
  `http://www.ibm.com/software/webservers/appserv/infocenter.html`
- CICS TS V2.2 online library
  `http://www-3.ibm.com/software/ts/cics/library/cicstsforzos22.html`
- WebSphere Application Server for OS/390 and z/OS library
  `http://www-3.ibm.com/software/webservers/appserv/zos_os390/library.html`
- WebSphere Application Server library
  `http://www.ibm.com/software/webservers/appserv/library.html`

  `http://www.ibm.com/support/techdocs`.
- IBM Technical Support Technical Information Site
  `http://www.ibm.com/support/techdocs`.

Referenced Web sites

These Web sites are also referenced as further information sources:

- https://www6.software.ibm.com/dl/websphere20/zosos390-p
  WebSphere Application Server for z/OS downloads
- http://www.verisign.com
  VeriSign, the Certificate Authority
- https://www6.software.ibm.com/dl/websphere20/zosos390-p
  WebSphere for z/OS and OS/390 tools
- http://www-3.ibm.com/software/webservers/appserv/zos_os390
  WebSphere application server home page
- http://www7b.boulder.ibm.com/wsdd/library/techarticles/0109_kelle/0109_kelle.html
  Whitepaper: *Using J2EE Resource Adapters in a Non-managed Environment*
- http://java.sun.com/j2ee/download.html#connectorspec
  J2EE connector architecture specification
- http://www.ibm.com/software/ts/cics/downloads
  CICS downloads page
- http://www6.software.ibm.com/dl/connarch/connarch-p
  Connector Architecture for WebSphere Application Server
- http://www.ibm.com/software/ad/studiointegration
  WebSphere Studio Application Developer Integration Edition
- http://www-4.ibm.com/software/ts/cics/announce
  CICS announcements
- http://www.ibm.com/support/techdocs
  IBM techdocs site
- http://www.dataset.fr/eng/scanport.html
  ScanPort download site

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

F

G

H

I

J

K

L

T

IBM

Redbooks

CICS Transaction Gateway V5: The WebSphere Connector for CICS

(0.5" spine)
0.475"<->0.875"
250 <-> 459 pages

CICS Transaction Gateway V5

The WebSphere Connector for CICS

Install and configure the CICS TG on z/OS, Linux, and Windows

Configure TCP/IP, TCP62, APPC or EXCI connections to CICS TS V2.2

Deploy J2EE applications to WebSphere Application Server V4 on z/OS or Windows

The CICS Transaction Gateway (CICS TG) is widely used to provide access to CICS COMMAREA-based programs and 3270 transactions from Java environments. This IBM Redbook shows you how to build a robust CICS TG configuration for a variety of different configurations.

First we introduce the facilities of the CICS TG, followed by step-by-step explanations of how to use the different protocols (TCP/IP, TCP62, APPC and EXCI) used for communication with a CICS TS V2.2 region on z/OS, and how to secure your CICS region when receiving External Call Interface (ECI) or External Presentation Interface (EPI) requests.

Next, we provide details on how to configure the CICS TG V5 on either z/OS or Linux to connect a Java client application to a CICS region. The use of the Secure Sockets Layer (SSL) protocol to encrypt the communication from the Java application to the CICS TG is included in these scenarios.

Finally, we offer two scenarios to illustrate how to configure WebSphere Application Server V4 on the Windows or z/OS platforms, to use the supplied ECI resource adapter to allow J2EE applications to make ECI calls to CICS.

SG24-6133-01 ISBN 0738426687