IBM® WebSphere®

The Complete Guide to CICS Transaction Gateway

Volume 1 Configuration and Administration

- A practical guide to the capabilities and usage of the CICS TG product suite
- Incremental setup from basic connectivity through to security and high availability
- Focus on IPIC connectivity, high availability, WebSphere Application Server, and CICS Explorer

Rufus Credle
Sue Bayliss
Leigh Compton
Robert Jones
Manuela Mandelli
Richard Mercadante

ibm.com/redbooks

Redbooks

International Technical Support Organization

IBM CICS Transaction Gateway Volume 1 Configuration and Administration

August 2014

SG24-8160-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (August 2014)

This edition applies to CICS Transaction Gateway for z/OS V9.0, CICS Transaction Gateway Desktop Edition V9.0, and CICS Transaction Gateway for Multiplatforms V9.0.

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 grant 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites 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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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 illustrate 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.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

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

AIX®
BladeCenter®
CICS®
CICS Explorer®
CICSPlex®
DB2®
Global Technology Services®
HiperSockets™
IBM®
Language Environment®
MVS™
OMEGAMON®
Parallel Sysplex®
Passport Advantage®
POWER®
Power Systems™
POWER7®
PureApplication®
RACF®
Rational®
Redbooks®
Redpaper™
Redpapers™
Redbooks (logo) ®
System i®
System x®
System z®
Tivoli®
TXSeries™
VTAM®
WebSphere®
z/OS®
z/VSE®
z10™

The following terms are trademarks of other companies:

Intel, Itanium, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

In this IBM® Redbooks® publication, you will gain an appreciation of the IBM CICS® Transaction Gateway (CICS TG) product suite, based on key criteria, such as capabilities, scalability, platform, CICS server support, application language support, and licensing model.

Matching the requirements to available infrastructure and hardware choices requires an appreciation of the choices available. In this book, you will gain an understanding of those choices, and will be capable of choosing the appropriate CICS connection protocol, APIs for the applications, and security options. You will understand the services available to the application developer when using a chosen protocol.

You will then learn about how to implement CICS TG solutions, taking advantage of the latest capabilities, such as IPIC connectivity, high availability, and Dynamic Server Selection. Specific scenarios illustrate the usage of CICS TG for IBM z/OS®, and CICS TG for Multiplatforms, with CICS Transaction Server for z/OS and IBM WebSphere® Application Server, including connections in CICS, configuring simple end-to-end connectivity (all platforms) with verification for remote and local mode applications, and adding security, XA support, and high availability.

Authors

This book was produced by a team of specialists from around the world working at IBM Hursley Park, United Kingdom.

Rufus Credle is a Certified Consulting IT Specialist at the ITSO, Raleigh Center. In his role as project leader, he conducts residencies and develops IBM Redbooks and Redpapers™ materials. Subjects include network operating systems, enterprise resource planning (ERP) solutions, voice technology, high availability, clustering solutions, web application servers, pervasive computing, IBM, and OEM e-business applications, WebSphere Commerce, IBM industry technology, System x®, and IBM BladeCenter®. Rufus' various positions during his IBM career include assignments in administration and asset management, systems engineering, sales and marketing, and IT services. He has a BS degree in Business Management from Saint Augustine's College. Rufus has been employed at IBM for 34 years.

Sue Bayliss is a Software Engineer in the UK. She has 20 years experience testing IBM software, including CICS Transaction Gateway, CICS Transaction Server, and WebSphere Application Server.

Leigh Compton is a Consulting IT Specialist with IBM Advanced Technical Skills in the US. She holds a degree in Education from Auburn University, along with graduate courses in Software Engineering and Information Science. She has 37 years of experience with CICS, and specializes in connectivity, integration, and security, along with CICS modernization.

Robert Jones works at the IBM Hursley Software Laboratory (United Kingdom), in the CICS strategy and technical planning team, with responsibility for the IBM CICS Transaction Gateway suite of products. He has worked on the development of CICS Transaction Gateway products since 2002, specializing on the IBM z/OS platform. He also produced the first CICS TG plug-in for IBM CICS Explorer®.

Manuela Mandelli is a Senior Software Specialist Professional and Certified Product Specialist Software for Technical Support Service of IBM Global Technology Services®. She is based in Milan, Italy.

She has 27 years of experience in supporting IBM Software products on mainframes. Her areas of expertise include CICS Transaction Server for z/OS, CICS Transaction Gateway, and IBM Session Manager.

She is a member of the CICS TS and CICS Tools Level 2 support team for Europe.

http://it.linkedin.com/pub/manuela-mandelli/58/286/51a

Richard Mercadante is a Technical Support Professional supporting both the CICS Transaction Gateway and IBM TXSeries™ CICS products.

He is based out of Pittsburgh, Pennsylvania in the US. He has over 35 years experience in the technology field with over 11 years supporting the CICS Transaction Gateway product.

He holds a degree in Administration and Management from La Roche College and a Certificate of Computer Systems Design from the University of Pittsburgh.

His areas of expertise include distributed operating systems and the CICS Transaction Gateway.

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

Tamikia Barrow-Lee, Richard Conway, Robert Haimowitz
International Technical Support Organization, Raleigh and Poughkeepsie Center

Geoff Pirie, Andy Bates, Clare Sprenger, Pete Dally, Colin Alcock, Kate Robinson, Steve Day, Phil Wakelin, Alan Hollingshead, Andy Murphy, Andrew Smithson
TP Development, IBM Hursley Park UK

Dario Facchinetti, Anna Defendi, Sergio Nolo
TSS, IBM Vimercate, Italy

Ted Caffarelli
IBM Buffalo USA

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

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 books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

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

  `ibm.com/redbooks`

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

Part 1

Introduction

By the end of Part 1, you will be able to choose the appropriate CICS Transaction Gateway (CICS TG) product based on key criteria, such as capabilities, scalability, platform, CICS server, application language support, and licensing model. You will also have a good understanding of terminology to be used throughout the book.

1

Basic concepts

This chapter describes the CICS Transaction Gateway (CICS TG) product family and provides positioning information for the individual products. It also introduces the main concepts for solution architects who require a broad awareness of the available capabilities.

The following topics are covered in this chapter:

1.1 Overview of the CICS TG products

The IBM CICS Transaction Gateway (CICS TG) product suite is the flexible, high-performing, security-rich, and scalable connector technology for all CICS servers. CICS TG-based solutions are production proven in client environments worldwide, with successful deployments across every major industry.

There are three CICS TG products available to suit a wide range of requirements (see Figure 1-1):

- IBM CICS Transaction Gateway for z/OS
- IBM CICS Transaction Gateway for Multiplatforms
- IBM CICS Transaction Gateway Desktop Edition

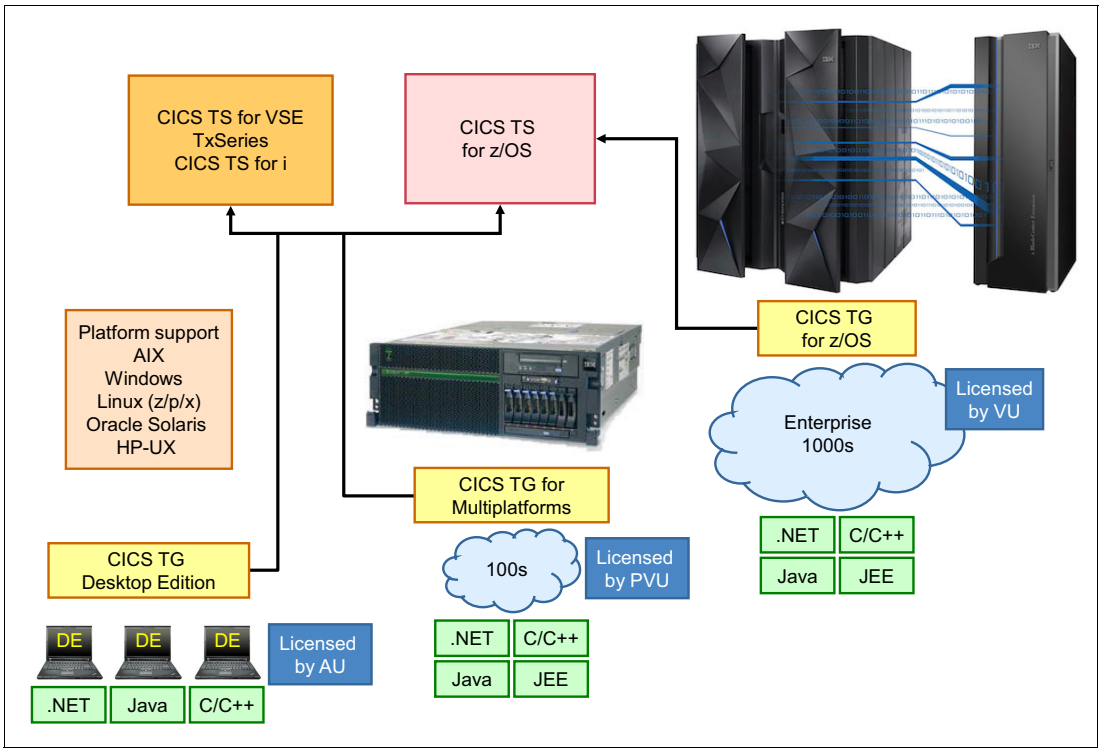


Figure 1-1 The IBM CICS Transaction Gateway products

The CICS TG products give application developers intuitive programming interfaces to access CICS business logic from multiple languages. They also use standard CICS intercommunication facilities, therefore requiring the minimal configuration changes to get started.

1.1.1 More you must know about IBM CICS Transaction Gateway

CICS TG products provide application integration in approximately half of all clients using CICS server products today, processing billions of transactions each day, worldwide.

Each CICS TG product enables integration for Java applications, servlets, or applets, .NET Framework-based applications, C, C++, and COBOL applications.

Both CICS TG for z/OS and CICS TG for Multiplatforms provide integration for Java Platform Enterprise Edition (Java EE)-certified application servers, with exclusive features for WebSphere Application Server products, such as identity propagation and cross-component trace (XCT). These two server-grade CICS TG products can be scaled to suit the capacity and availability requirements of the application workload. The CICS TG Desktop Edition product is licensed for single users. It is aimed at providing direct connectivity to CICS servers for desktop applications.

Running on IBM z/OS and Microsoft Windows, Linux, and UNIX operating systems, CICS TG products provide connectivity to all IBM CICS server products:

- CICS Transaction Server for z/OS
- TXSeries for Multiplatforms
- CICS Transaction Server for IBM z/VSE®
- CICS Transaction Server for i

1.1.2 Business value

CICS TG products deliver proven integration technology and have a long track record of reliable and robust connectivity for all CICS server products. Application developers use standard tools, such as Eclipse, the IBM Rational® Application Developer family, and Microsoft Visual Studio. System administrators use standard facilities on their platform of choice for installation, maintenance, and configuration.

CICS TG products use industry-standard transports and standard CICS intercommunication facilities, offering maximum flexibility to adapt solutions as business requirements evolve over time. As networking technology advances, CICS TG products transparently reap benefits delivered through the TCP/IP and secure socket libraries. As server hardware and operating systems advance, CICS TG components and applications automatically benefit through optimizations in compilers and Java run time. Examples include EC12 processor optimizations, large page support with 64-bit Java and zAAP offload on z/OS, or IBM POWER7® processor optimizations on IBM AIX®.

CICS TG application programming interfaces (APIs) provide developers with intuitive access to CICS business logic and data. Portability of application code across hardware and operating systems delivers longevity for the initial investment as platform technologies evolve. For example, application code originally developed for use with CICS TG for Multiplatforms on UNIX or Windows might be reused in a larger scale deployment with CICS TG for z/OS. In a three-tier solution, applications developed for different runtime environments can share access to CICS servers through the same Gateway daemon.

The Java EE-certified CICS connectors offer flexibility when choosing a Java EE application server vendor, and fully uses unique features of the IBM WebSphere Application Server products. Java EE applications deployed to IBM PureApplication® System integrate with CICS business logic and data through CICS TG, using the Web Application pattern.

CICS TG for z/OS uses the unique capabilities offered by z/OS, taking advantage of the qualities of service and scalability offered by IBM System z® IBM Parallel Sysplex®. Support for industry-standard hypervisors also enables deployment of CICS TG for Multiplatforms to virtualized servers, including Linux for System z and workload partitions (WPARs) on IBM AIX.

1.1.3 Solution overview

This topic summarizes the capabilities provided by the CICS Transaction Gateway products.

CICS TG programming interfaces provide a tight coupling between applications, although the CICS TG runtime components provide scalability, availability, and security for the solution. Tight coupling is appropriate when there is a well-defined relationship between the CICS business logic and the non-CICS applications. However, Web Services are more appropriate when a loose coupling is required. Loose coupling is appropriate for exposing business logic when the partner system is unspecified.

All CICS TG products provide the following APIs for multiple programming languages:

- External Call Interface (ECI) provides access to CICS programs, with data encapsulated as *COMMAREAs* or *channels and containers*
- External security interface (ESI) provides access to Password Expiry Management (PEM) capabilities, such as verifying a password or changing a password

The CICS TG for Multiplatforms and CICS TG Desktop Edition products additionally provide the external presentation interface (EPI) for access to CICS terminal transactions through virtual 3270 terminals. Integration with CICS Transaction Server for z/OS (CICS TS), or CICS TS for z/VSE terminal-based transactions requires a Systems Network Architecture (SNA) Advanced Program-to-Program Communication (APPC) connection, provided through an additional product, such as IBM Communications Server for Data Center Deployment or IBM Host Access Client Package. EPI also offers integration with CICS TS for i terminal-based transactions using either a `TCPIP` or `SNA` (APPC) connection, and with TXSeries terminal-based transactions using a TCPIP connection.

1.1.4 Solution architecture

There are three models of deployment for CICS TG applications, where the location of the application relative to the CICS TG product installation influences which APIs an application might use. Figure 1-2 on page 7 shows the CICS Transaction Gateway architecture options.

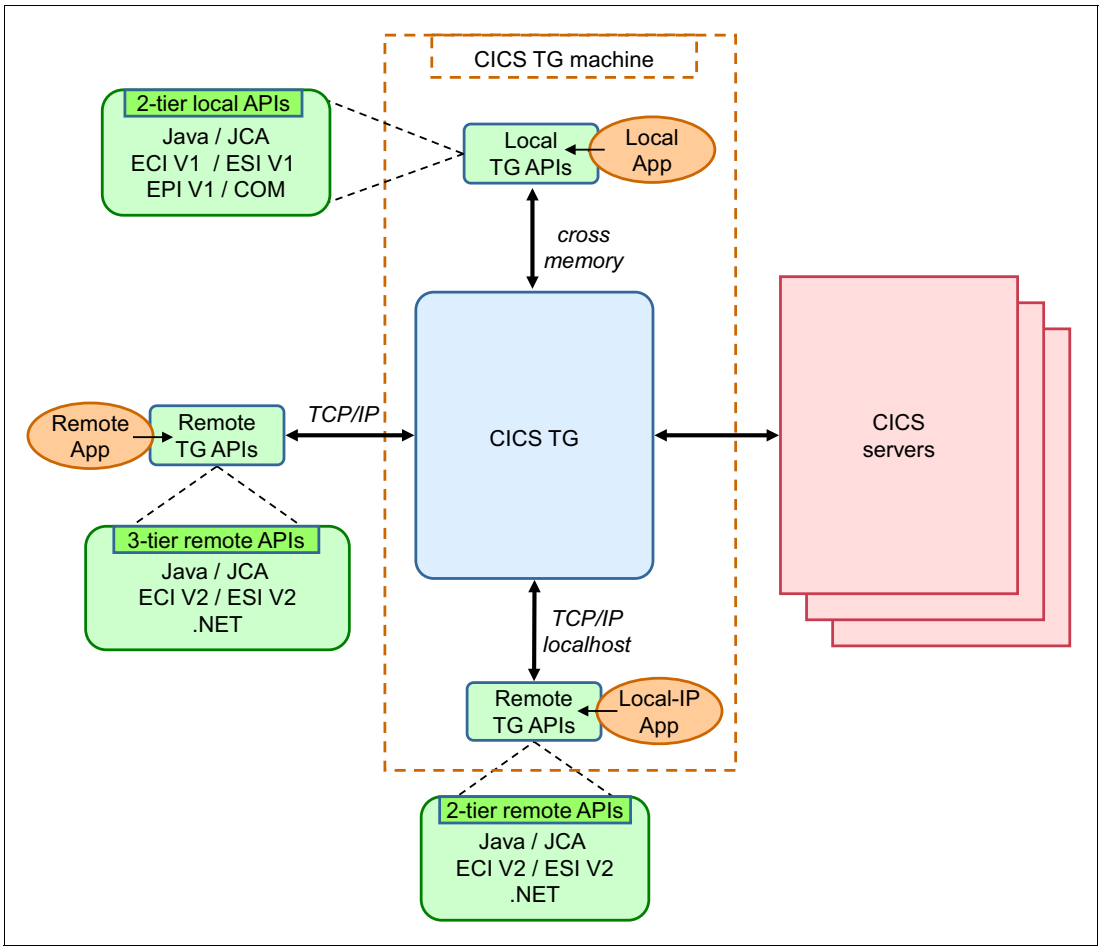


Figure 1-2 CICS Transaction Gateway solution architecture options

The three models of deployment can be summarized as noted in the following list:

- Three-tier remote mode:
  - Application is on a separate machine from the CICS TG product installation.
  - Application requests flow to CICS through a remote Gateway daemon component.
- Two-tier remote mode:
  - Application is colocated with the CICS TG product installation.
  - Application requests flow to CICS through a colocated Gateway daemon.
- Two-tier local mode:
  - Application is colocated with the CICS TG product installation.
  - Application requests flow directly to the CICS server.

Two-tier local mode represents the shortest end-to-end path length to process an application request. However, 3-tier remote mode provides the greater qualities of service in terms of flexibility, availability, and scalability.

Although CICS TG Desktop Edition includes the remote mode API components for Java, ECI V2, ESI V2, and .NET Framework-based applications, the product license restricts their use to applications colocated with the product installation.

1.1.5 Usage scenarios

The following scenarios describe actual client deployments, where CICS TG was selected as the integration technology from a range of possible solutions. The flexibility offered by the CICS TG products is often cited as a winning factor when selecting integration middleware for CICS.

Retail banking

An expanding international banking group had a branch-based application founded directly upon SNA network protocols to provide communications with a central mainframe. To apply modern levels of security, improve overall stability, and establish a scalable, robust, and extensible service, they decided to migrate to a TCP/IP-based middleware technology.

After evaluating a range of options, they chose to use CICS Transaction Gateway on z/OS, connecting each Windows desktop through the TCP/IP network using the CICS TG, ECI Version 2, API. The solution included eight Gateway daemons running on four z/OS logical partitions (LPARs) (two Gateway daemons per LPAR) to provide the required capacity and availability.

Within six months of the project starting, the new solution helped process nearly 15 million transactions per day. It now connects more than 3,000 branches with approximately 10 Windows desktop machines per branch. IBM CICS Transaction Gateway software has helped the banking group to deliver a stable, secure, and high-performance transaction processing solution for a significant part of its business. The success of this project has made this solution a strong contender for future projects within the group.

Financial services

A key financial services provider for multiple banks in a leading European economy recognized the need to transform its core services to serve a rapidly expanding market. It recognized that sustained growth requires a standard but highly scalable solution for transforming 3270-based applications for web enablement. The solution must be flexible enough to accommodate new applications without disrupting existing applications and cope with significant increases in workload within relatively short periods.

CICS Transaction Gateway enabled a smooth transformation to the use of modern Java Enterprise Edition applications, running on WebSphere Application Server and integrated with CICS Transaction Server for z/OS. CICS TG provides flexible configuration options, supporting a mixture of application and system architectures and allowing use by multiple projects. CICS TG provides high-performing and scalable access to proven CICS business logic, requiring minimal changes to CICS systems and applications. The final solution reduced costs, risks, and time-to-market by the increased reuse of applications and skills (see Figure 1-3 on page 9).

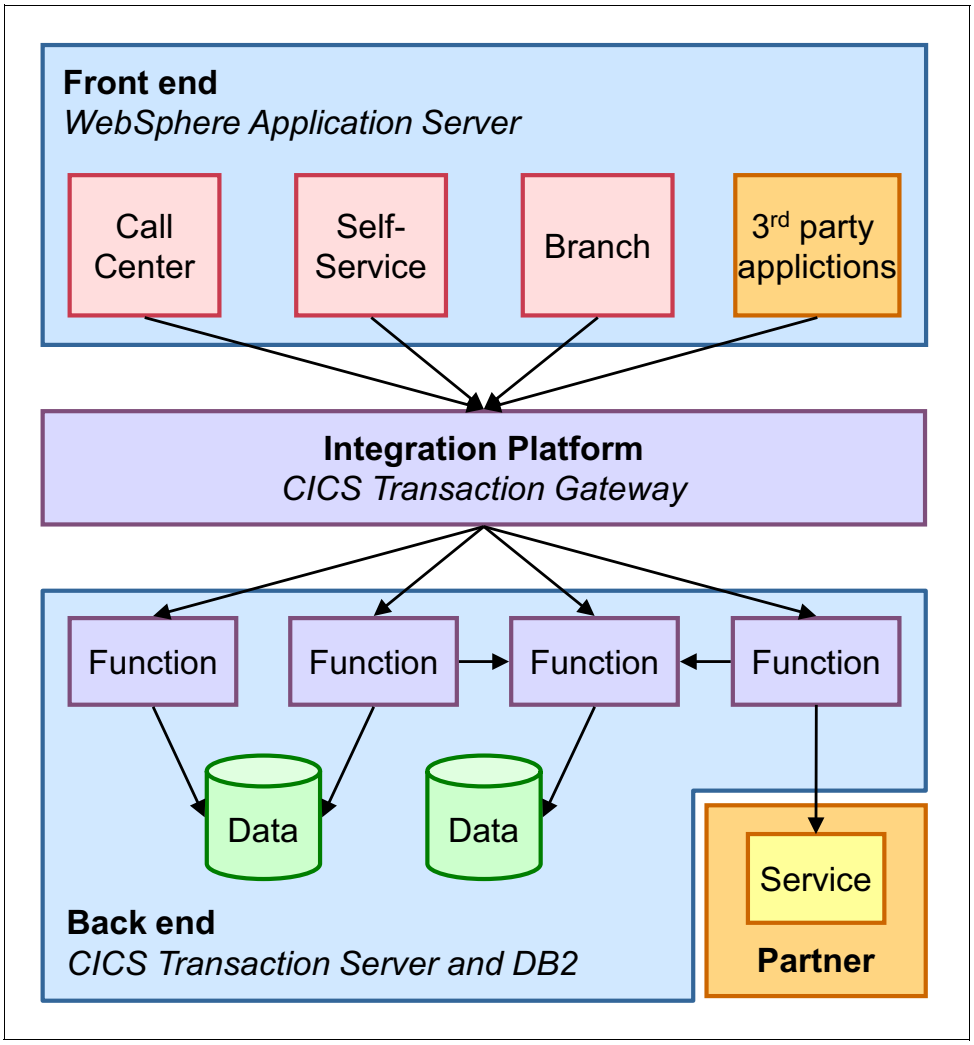


Figure 1-3 Scalable WebSphere Application Server integration solution

The workload handled by the new system increased by 300% over the following two years. The peak number of transactions processed through their CICS TG integration platform in one day reached 12.5 million. Workloads through this system peak at between 120 and 130 transactions per second, while maintaining response times below 0.1 seconds.

This client has also deployed CICS TG for z/OS in a similar pattern for other solutions, the largest of which processes more than 200 transactions per second. Overall, the company process more than 50 billion transactions each year, with peak rates of 250 million transactions in one day and 30 million transactions in one hour.

1.1.6 Integration

CICS Transaction Gateway integrates with a wide selection of IBM software:

- Middleware products, such as WebSphere Message Broker and WebSphere Application Server
- Performance tuning and monitoring tools, such as CICS Performance Analyzer and IBM Tivoli® OMEGAMON® XE for CICS on z/OS
- Application development and testing products such as Rational Application Developer and Rational Integration Tester

Figure 1-4 shows a summary of various product integrations.

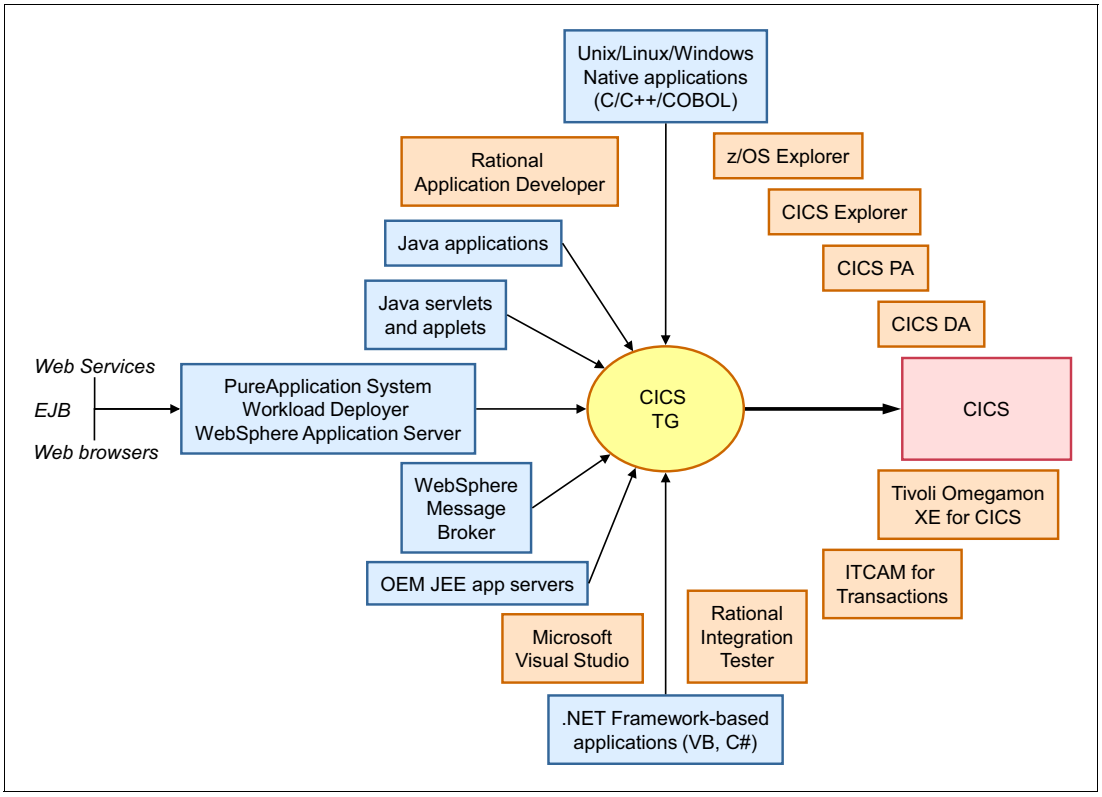


Figure 1-4 CICS Transaction Gateway product integration summary

CICS portfolio

CICS Performance Analyzer (CICS PA) takes System Management Facility (SMF) type 111 records, generated by CICS TG for z/OS, and offers a powerful range of options for summarizing large sets of historical statistics, potentially generated by multiple Gateway daemons. Predefined reports available from CICS PA are included in the following list:

- Activity Summary
- Usage and Capacity
- Configuration Summary
- Client Workload
- CICS Workload

Custom reports can be defined based on conditions that are defined in terms of CICS Transaction Gateway statistics and threshold values.

CICS Deployment Assistant (CICS DA) is able to discover and visualize Gateway daemon address spaces running anywhere within the sysplex, and shows CICS TG in a wider context of CICS, IBM DB2®, and WebSphere MQ resources.

CICS Explorer (and z/OS Explorer) can use the CICS TG plug-in to provide visibility of multiple Gateway daemons running as part of any CICS TG product on any platform. A system administrator has live access to many runtime attributes for each Gateway daemon and its associated CICS connections, and the ability to actively verify basic connectivity.

WebSphere products

CICS TG for z/OS and CICS TG for Multiplatforms provide the CICS resource adapters for use with the WebSphere Application Server family of products. They allow Java EE application developers to use CICS resources through the standard JCA interfaces.

The ECI resource adapter represents the most popular means of CICS integration for Java EE application developers, providing access to CICS business logic and data through a standard CICS program link. It also provides options on transactional behavior including distributed two-phase commit, based on the eXtended Architecture (XA) specification, from The Open Group. Combined with CICS TG for z/OS in a high availability configuration, XA transaction support delivers a robust, scalable solution maximizing the highest qualities of service available.

The EPI resource adapter enables Java EE application developers targeting WebSphere Application Server for Multiplatforms to access terminal-based transactions on any CICS server.

WebSphere Message Broker includes a *CICS node* enabling integration with a remote-mode CICS Transaction Gateway configuration. This allows WebSphere Message Broker to immediately use established CICS TG solutions, or build scalable and highly available CICS connectivity through a new CICS TG solution.

IBM Integration Designer is an Eclipse-based software development tool that renders your current IT assets into service components for reuse in service-oriented architecture (SOA) solutions. IBM Integration Designer bundles the CICS TG for Multiplatforms product, which is licensed for application development and test purposes.

PureApplication System

CICS TG enables the integration of Java EE applications deployed to a private cloud with CICS servers outside the cloud. IBM PureApplication System provides Virtual application patterns, which include Virtual application pattern components. Transaction processing components include the *Existing CICS Transaction Gateway* component, allowing Java EE applications deployed within the cloud to access CICS business logic through an existing Gateway daemon, running outside the cloud.

The Existing CICS Transaction Gateway component is available within the IBM Web Application Pattern, which provides a set of components that are typical for web applications. After building the virtual application in the Virtual Application Builder, you can deploy the application and the system determines the underlying topology configuration.

The Existing CICS Transaction Gateway component is also included in the Virtual application builder of IBM Workload Deployer appliance.

Tivoli products

There are two products in the IBM Tivoli Monitoring portfolio that monitor CICS TG to help with problem identification, resolution, and prevention:

- IBM Tivoli OMEGAMON XE for CICS on z/OS delivers a range of capabilities for the z/OS or CICS system administrators:
  - In-depth monitoring of CICS TG instances running on z/OS
  - Remote mode and WebSphere Application Server for z/OS applications
  - Alerts for communication problems between CICS TG for z/OS and CICS TS for z/OS
  - Up-to-the-second details about transaction rate, I/O, and CPU consumption
  - Trend analysis to assist with future capacity planning
  - Easy access to monitoring tools of other System z applications
- IBM Tivoli Composite Application Manager for Transactions provides transaction tracking and visualization capabilities. It helps to identify transaction response and availability issues, enabling faster problem resolution through the following features:
  - End-to-end analysis of transaction response time and performance
  - Full transaction tracking of flows through CICS TG on both z/OS and Multiplatforms
  - Identification of connections from WebSphere through CICS TG to CICS (and onward)
  - Automatic identification for points of slowdown and deviation from typical response times

Rational products

IBM Rational Application Developer includes CICS TG for Multiplatforms, licensed for application development and test purposes. The Java Platform 2, Enterprise Edition Connector architecture tooling also provides wizards to assist with the generation of Java classes from COBOL copybooks, enabling Java application developers to work intuitively with data structures originating from CICS COBOL business logic.

Rational Integration Tester, a component of IBM Rational Test Automation Solution, includes a code-free solution for continuous integration testing of CICS programs through CICS TG. Conversely, it also provides the ability to virtualize CICS server responses, enabling continuous integration testing of Java or JCA applications using the ECI API. Record and replay of ECI requests further facilitate the building of test cases based on real-world data, or automation of existing test cases.

1.1.7 Supported platforms

Each CICS TG product includes several runtime components that are supported on *Multiplatforms*. Within the context of the CICS TG products, the term *Multiplatforms* broadly refers to these operating systems and architectures:

- AIX (IBM POWER®)
- Windows (Intel)
- Red Hat Enterprise Linux (Intel, POWER, and System z)
- Red Hat Enterprise Linux-compatible distributions (Intel)
- Red Hat Enterprise Desktop (Intel)
- SUSE Linux Enterprise Server (Intel, POWER, and System z)
- SUSE Linux Enterprise Desktop (Intel)
- Solaris (SPARC)
- HP-UX (Itanium)

The specific releases of supported operating systems and hardware architectures are constantly evolving over time and vary accordingly with each new release of the CICS TG products. So, this list represents a high-level summary.

Each release of CICS TG for z/OS requires a specific minimum release of z/OS, and is typically supported on subsequent releases of z/OS unless explicitly stated otherwise by IBM.

CICS TG for Multiplatforms and CICS TG Desktop Edition support the Multiplatforms operating systems and architectures, but are also subject to licensing terms and conditions. For example, installing CICS TG Desktop Edition on a UNIX or Linux workstation is supported for a single user, but not for a UNIX or Linux server machine supporting multiple concurrent users.

CICS TG products typically support all in-support releases of all CICS server products, including the following list:

- CICS Transaction Server for z/OS
- TXSeries for Multiplatforms
- CICS Transaction Server for z/VSE
- CICS Transaction Server for i

Each CICS TG product includes runtime components for use by Java applications with supported releases of Java Standard Edition (SE). In addition to IBM-supplied Java runtime environments, any certified Java-compatible Java SE Runtime Environment is also supported. By contrast, the Gateway daemon must run under an IBM supplied Java runtime environment of a specified release, for a given CICS TG product and release.

Both CICS TG for z/OS and CICS TG for Multiplatforms products include JCA resource adapters for use with Java EE-certified application server products, such as IBM WebSphere Application Server. Java EE-certified application server products, other than WebSphere Application Server, must successfully install and run the CICS TG JCA installation verification and test (`IVT`) program to gain entitlement to IBM support.

Both CICS TG for z/OS and CICS TG for Multiplatforms products include redistributable runtime libraries suitable for deployment on any of these Multiplatforms operating systems. This allows many types of remote applications to connect through a Gateway daemon, no matter which server platform is used.

The web page, Supported Software for CICS Transaction Gateway (CICS TG) products, provides links to the software requirements information center topic for all in-support releases of each CICS TG product. Each information center also includes a hardware requirements topic. Visit the following link for more information:

http://www.ibm.com/support/docview.wss?uid=swg21239203

1.2 Connectivity and capability by CICS TG product

Selecting the appropriate CICS TG product can be influenced by many competing characteristics of the overall solution. There are variations in the capabilities of the CICS TG products. Within each product, there is a direct relation between the APIs available for use by an application, and the available CICS connection protocols.

Table 1-1 on page 14 summarizes the capabilities and options for connectivity between combinations of CICS TG and CICS server products.

Table 1-1 Capabilities for CICS TG and CICS server products

CICS protocol	Available with CICS TG products	Connects with CICS server products	Used with CICS TG APIs
IPIC	z/OS Multiplatforms Desktop Edition	CICS TS for z/OS TXSeries for Multiplatforms	ECI COMMAREA and Channels ESI (CICS TS for z/OS only)
EXCI	z/OS	CICS TS for z/OS	ECI COMMAREA
TCP/IP	Multiplatforms Desktop Edition	CICS TS for z/OS CICS TS for z/VSE TXSeries for Multiplatforms CICS TS for i	ECI COMMAREA EPI (TXSeries for Multiplatforms and CICS TS for i)
SNA APPC	Multiplatforms Desktop Edition	CICS TS for z/OS CICS TS for z/VSE CICS TS for i	ECI COMMAREA EPI ESI

1.3 CICS TG topologies

The CICS Transaction Gateway products provide a highly flexible approach to integrating CICS servers with many types of applications. When selecting the topology to use for a particular application, the solution architect needs to consider where the application will run in relation to where the CICS TG product is installed, together with the qualities of service required to meet the requirements of the user.

For CICS TG for z/OS and CICS TG for Multiplatforms, the fundamental choice lies in using a 3-tier or 2-tier solution. The CICS TG Desktop Edition product license restricts usage to 2-tier solutions.

1.3.1 Three-tier solutions

In a 3-tier solution, an application uses the CICS TG API components to interact with a CICS server through an intermediary component, the Gateway daemon. This type of solution is known as *remote mode* in the CICS TG reference material (and this book), because the application runs on a machine that is remote to where the CICS TG product is installed.

A remote mode configuration uses two network links: one link from the CICS TG API components to the Gateway daemon, and a second link from the Gateway daemon to the CICS server. The Gateway daemon runs on the machine where the CICS TG product is installed. The CICS TG API components required by the application can be redistributed and have a relatively small footprint compared to the CICS TG product.

A 3-tier solution offers a wide range of capabilities to an overall solution including better scalability, availability, flexibility, monitoring, and statistics. The additional level of indirection enables the middleware to bring more value to the solution over time without changes to application code. Also, it effectively removes affinities between application code and CICS server infrastructure, allowing for transparent operation through reorganization of CICS servers.

1.3.2 Two-tier solutions

In a 2-tier solution, an application uses the CICS TG API components to interact directly with a CICS server. This type of solution is known as *local mode*, because the application runs on a machine where a CICS TG product is installed.

A local mode configuration uses a single network link from the CICS TG product to the CICS server. The CICS TG API components use the facilities of the local CICS TG product directly.

There is a special case for 2-tier solutions, where the application runs on the machine where the CICS TG product is installed, but connects through the Gateway daemon (using the local network adapter). This might be necessary when a local application uses one of the CICS TG APIs requiring a Gateway daemon connection, such as .NET, ECI Version 2, or ESI Version 2. This style of configuration allows the use of these remote APIs with the CICS TG Desktop Edition product.

1.3.3 CICS TG for z/OS

CICS TG for z/OS is typically installed within the same LPAR as CICS TS for z/OS. This provides the Gateway daemon with a low-latency link to CICS servers for remote mode applications connecting through the Gateway daemon, or local mode applications deployed on a colocated WebSphere Application Server for z/OS node.

WebSphere Application Server for z/OS is the primary use case for local mode operating with CICS TG for z/OS. However, WebSphere Application Server for z/OS can be used with a colocated Gateway daemon (2-tier remote), for example, to use high availability features.

The CICS TG for z/OS pricing model is reference-based on the CICS workload, and is measured in Value Units. If a given workload is processed by a single Gateway daemon, or distributed between multiple Gateway daemons each, the overall cost is comparable. For this reason, it is common to use CICS TG for z/OS in remote mode with multiple Gateway daemons sharing a common workload. Each Gateway daemon typically runs below maximum capacity but is able to take up additional load if required.

When an IPIC connection is made between a Gateway daemon and a CICS region or WebSphere Application Server and a CICS region within the same LPAR, then IBM Communications Server for z/OS can optimize that connection using Fast Local Sockets. Fast Local Sockets maximize cross-memory facilities between address spaces within the same z/OS LPAR to provide optimized TCP/IP connections.

CICS TG for z/OS is best suited for integration with CICS TS for z/OS, but IPIC connections also offer the option to integrate with TXSeries for Multiplatforms.

1.3.4 CICS TG for Multiplatforms

CICS TG for Multiplatforms is installed on Windows, UNIX, or Linux servers, including Linux for System z, and offers a high degree of flexibility for deployment options and integration.

It can be used in both 3-tier solutions and 2-tier solutions, integrating with all supported CICS server products. Although it can be scaled through deployment to multiple servers, the pricing model differs from the z/OS product in that each server is licensed separately.

CICS TG for Multiplatforms can use TCP/IP-based connectivity and SNA-based connectivity, depending on the details of the required integration. However, SNA APPC connections require an additional product to provide the underlying SNA communications layer.

CICS TG for Multiplatforms provides the widest range of CICS TG APIs, including Java EE resource adapters for Java EE application servers, such as WebSphere Application Server for Multiplatforms, and terminal-based transactions through EPI.

1.3.5 CICS TG Desktop Edition

CICS TG Desktop Edition is installed on individual user machines, and is licensed on a per-user basis. It provides all of the APIs available with CICS TG for Multiplatforms, except the JCA resource adapters, and can only be used in 2-tier solutions. It is capable of supporting the workload of a single user, and scalability is achieved through the installation of the product and application to multiple user machines.

CICS TG Desktop Edition provides an upgrade path for users of the discontinued product, IBM CICS Universal Client, bringing support for modern compilers and operating systems. It includes the features of that product, but also adds more APIs: Java base classes, ECI V2, ESI V2, and .NET Framework-based APIs.

1.4 High availability features

Each of the CICS TG products includes capabilities to maintain the availability of the link between the application and the CICS servers. However, as the CICS TG products have evolved and new capabilities have been added, older techniques have been superseded but retained for compatibility.

1.4.1 Connection balancing

Remote mode solutions can be scaled by creating multiple Gateway daemons and balancing connections from applications between them. This can be achieved on z/OS through a combination of port sharing within a z/OS LPAR and connection balancing between LPARs, using Sysplex Distributor, or an external device. For Multiplatforms, connections can also be balanced between multiple Gateway daemons running on physically separate machines, or on virtualized server instances.

In both cases, the application workload connects to a *logical Gateway daemon*, which might consist of any number of real Gateway daemon instances all providing access to equivalent business logic and data.

1.4.2 Dynamic Server Selection

Traditional CICS TG applications specify a CICS server name in the request, which must resolve to an actual CICS server at run time. For the CICS TG products, Dynamic Server Selection refers to the process of modifying the target CICS server at run time, based on some predetermined algorithm. This allows the CICS server infrastructure and application logic to evolve independently, with dynamic routing of requests and transparent failover.

Modern Dynamic Server Selection is available for remote mode solutions, through a configurable policy with CICS TG for z/OS, or through a Gateway daemon user-exit, available for each CICS TG product.

Although older techniques are still available, only the Gateway daemon-based Dynamic Server Selection functions interoperate with all CICS connection protocols, including IPIC, channel and container-based ECI requests, XA transactions, request monitoring exits, and CICS TG statistics.

1.5 Transactionality

CICS TG application programmers must consider the types and effects of different transaction models associated with the APIs that they are using. EPI requests invoke CICS terminal transactions by transaction ID. By their nature, such requests are already limited to committing or backing out updates to recoverable resources when control is returned to the terminal. By contrast, ECI requests can be made in isolation, or in composite into a single transaction.

This topic summarizes the transactional models available for use with ECI applications, including considerations for CICS TG products, CICS server products, CICS connections, and CICS TG APIs. For greater detail, see the explanation in 6.3, “Transactional models” on page 97.

1.5.1 Sync-on-return

When an ECI application invokes business logic on a CICS server, it can result in one or more updates to recoverable resources. When the CICS business logic represents a self-contained operation, any updates made to recoverable resources will be synchronized when the operation ends. This is known as *sync-on-return*, because CICS performs a sync point on involved recoverable resources, at the point of return.

Sync-on-return is the most simple and commonly used transaction model with CICS TG products. However, it is not sufficient when the application needs to synchronize updates to recoverable resources resulting from multiple ECI calls.

Each of the CICS TG products supports ECI sync-on-return, for all of the supported programming languages, topologies, CICS servers, and runtime environments.

1.5.2 Distributed Transaction Processing

When a sequence of CICS business logic invocations is combined to form a composite operation, and the composite operation must be atomic, distributed transaction processing (DTP) is used to coordinate synchronization of recoverable resources.

The method of DTP employed for a particular solution depends on the specific CICS TG product, topology, the type of CICS servers involved, CICS communications protocol, the CICS TG API, and the application runtime environment.

Extended logical unit of work

Extended logical units of work (LUW) provide a single-phase commit model of DTP, allowing multiple ECI calls to be enrolled in a single transaction, before the application logic decides to commit or roll back. Single phase commit carries the risk of an uncertain outcome in the event of a failure, such as a communications failure, or timeout.

Each of the CICS TG products supports ECI extended LUWs for all of the supported programming languages, topologies, CICS servers, and runtime environments.

XA transactions

XA transactions deliver a full two-phase commit model of DTP, delivering full transactional integrity. Two-phase commit avoids the need for application logic to compensate for potential inconsistencies following an unexpected condition in the overall system.

XA transaction support is available with the CICS TG for z/OS and CICS TG for Multiplatforms products, and is limited to JCA applications deployed to a Java EE-certified application server using the ECI resource adapter.

The CICS TG for z/OS product supports ECI XA transactions in local mode (WebSphere Application Server for z/OS), and remote mode. Sysplex-wide high availability configurations are supported including use of Dynamic Server Selection. Both IPIC and EXCI connections to CICS TS can be used with XA transactions, and Resource Recovery Services (RRS) underpins the overall solution for full sysplex utilization.

The CICS TG for Multiplatforms supports ECI XA transactions in local mode from a Java EE-certified application server, using IPIC connections to CICS TS for z/OS V3.2, or later.

Out of all the CICS server products, only CICS TS for z/OS includes support for XA transactions.

1.6 Security

There are many security features available for use with CICS TG-based solutions. These range from unencrypted unauthenticated, to fully encrypted and secure authentication through the latest cipher suites.

All remote mode applications can connect to a Gateway daemon using a non-secure TCP/IP-based connection. Remote mode applications based on the Java base classes, JCA, and .NET Framework-based APIs can also establish a secure connection with the Gateway daemon. Secure connections use server authentication, and can optionally require client authentication.

CICS server connections are established by CICS TG runtime components from the middle tier (remote mode) or the first tier (local mode), using one of the protocols listed in Table 1-1 on page 14. By default, these connections use non-secure network transports, or a cross-memory-based transport in EXCI. Although cross-memory interfaces can be regarded as being implicitly secure, only IPIC connections can be directly configured to use modern standards of encryption and authentication.

CICS TG application credentials are traditionally based on a user ID and password that must be authenticated successfully before the application request can be processed. More secure credentials include password phrases, and Secure Sockets Layer (SSL) with client authentication.

Identity assertion is a common security pattern requiring a pre-authenticated user ID as a credential. With CICS TS for z/OS, identity assertion is possible when the connection is *trusted*. The state of trust depends on either the relative location of the partner system, or the type of connection used:

- Connection is IPIC SSL from a local mode application or Gateway daemon, on any platform
- Connection is IPIC and originates from a z/OS LPAR within the same sysplex
- Connection is EXCI

> **Note:** A Linux on System z LPAR cannot be regarded as a trusted partner based on its location. Instead, an IPIC SSL connection must be used as the basis for trust.

With TXSeries, an IPIC connection can be defined to accept pre-authenticated requests by setting the RemoteSysSecurity attribute to "`trusted`", in the `CD.stanza` file.

For full details about security options, see Chapter 8, "Security" on page 131.

2

CICS Transaction Gateway for z/OS installations

This chapter describes the file structures of CICS Transaction Gateway (CICS TG) on z/OS, as the result of the SMP/E for z/OS installation.

It also introduces several fundamental concepts related to the structure of CICS TG for z/OS to better identify the required z/OS and CICS TG elements to implement the best solution, based on key criteria, such as capabilities and scalability. The following concepts are covered in this chapter:

2.1 Installing CICS TG for z/OS

This section explains how to manage all the z/OS and CICS Transaction Gateway elements for controlling an installation.

2.1.1 Ordering CICS TG from Shopz

CICS TG can be ordered using Shopz. It can be in ServerPac or IBM Custom-Build Product Delivery Offering (CBPDO) format. Shopz will analyze your current installation, determine the correct product migration, and present your new configuration based on z/OS.

Shopz is an Internet application that you can use to order z/OS software products and services. Using Shopz, you can order corrective and preventive service over the Internet, with delivery over the Internet or by tape. The ordering service with Shopz reduces your research time and effort by using your uploaded SMP/E consolidated software inventory (CSI) to ensure that all applicable services, including reach-ahead service, for the installed Function Module IDs (FMIDs) in the target zones are selected. The Shopz web address is listed:

http://www.ibm.com/software/shopzseries

Shopz provides entitled service ordering and service delivery capabilities for the z/OS platform products electronically using the Internet. Shopz is the primary ordering and delivery method for software service for ordering and receiving z/OS.

2.2 Anatomy of CICS TG for z/OS

Whatever format you use to perform the CICS TG installation, the result of the SMP/E installation creates ISPF data sets and files in zFS. These data set files and z/FS files are used to run CICS TG.

All CICS TG installations create the IBM MVS™ data sets. The content of the MVS data sets is described in Table 2-1.

Table 2-1 z/OS data set list

Data set name	Member name	Description
hlq.SCTGAUTH	CTGARM	Authorized program facility (APF)-authorization modules
hlq.SCTGDLL	CTGSTATS	Dynamic link library (DLL) modules
hlq.SCTGHFS	*VSAM file	To create and mount the z/OS UNIX files
hlq.SCTGINCL	CTGSMF CTGSTATS CTGSTDAT	The C header files for System Management Facility (SMF)

Data set name	Member name	Description
hlq.SCTGINST	CTGACCPT CTGALLOC CTGAPPLY CTGDDDEF CTGISMKD CTGIZFS0 CTGIZFS1 CTGMKDIR CTGRECV CTGRECVE CTGSMPSU	SMP/E installation jobs
hlq.SCTGLINK	CTGINIT	LNKLST member
hlq.SCTGLOAD	CTGBATCH	CICS TG load modules
hlq.SCTGMAC	CTGSMFA	z/OS assembler macro for SMF formatting job
hlq.SCTGSAMP	CTGASIS CTGCONF CTGCONV CTGENV CTGJOB CTGPROC CTGSMFB CTGSMFR CTGSMFRD CTGSSLPW CTGSTAT1 CTGSTJOB CTGS01A1 CTGS02A1 CTGS03A1 CTGS03A2 CTGS04A1 CTGS05NV CTGS08A1 CTGTESTL CTGTESTR	Samples and scenarios information
hlq.SCTGSID	CTGSTATS	Statistic information

The data set name must be a unique name. It is built by a different level of qualifier. The first part of the name is called the high-level qualifier (hlq). The standard convention identifies the hlq with a name that allows you to refer to a particular version, release, and modification of a specific product, in our example, the hlq is `CICSTG.V9R0M0`.

A suggested place to start your CICS TG installation is by using the samples provided by the `SCTGSAMP` library. MVS data set members `CTGCONF` and `CTGENV` are the configuration files. They can be used to start CICS TG. You can create the startup job using **`CTGJOB`** or **`CTGPROC`**, based on whether you want to start CICS TG as a batch job or a started procedure.

`SCTGSAMP` also contains scenarios that show how to perform tasks, such as configuring connections to CICS, or configuring Secure Sockets Layer (SSL) security. For a detailed description of the scenarios, see the CICS TG Information Center:

http://ibm.co/1tfLphd

Table 2-2 describes the content of the files created in the z/OS UNIX environment.

Table 2-2 z/OS UNIX files description

z/OS UNIX directories	Subdirectories	Description
bin	resource	Executable binaries
classes		CICS TG Java classes
deployable		Deployable EAR and RAR files and `ctgredist.tar.gz` containing the redistributable components for developing both C and .NET applications
docs		CICS TG API reference documentation
IBM		
msgs		
samples	configuration Java scenarios server	CICS TG samples and scenarios files

The directory name will be prefixed by your installation directory. The default is `/usr/lpp/cicstg/ctg900`, where the last subdirectory, for example, `ctg900`, refers to a particular version, release, and modification of the product.

The `deployable` directory contains `cicseci.rar` (the CICS ECI resource adapter), which provides one-phase and two-phase transaction support when installed into the Application Server. The `ctgredist.tar.gz` contains the `rar` file for .NET and assembler.

The `sample` directory contains sample Java programs and configuration files to set up the scenarios described previously.

2.3 CICS TG basic structure

This section describes the relationship between the CICS TG for z/OS components and the z/OS system, when the remote mode topology is used. The local mode topology will be described later in this book.

This section will provide the CICS TG basic component descriptions to support a better understanding of the CICS TG anatomy and how it interacts with a z/OS system.

Figure 2-1 on page 25 shows the main CICS TG runtime components.

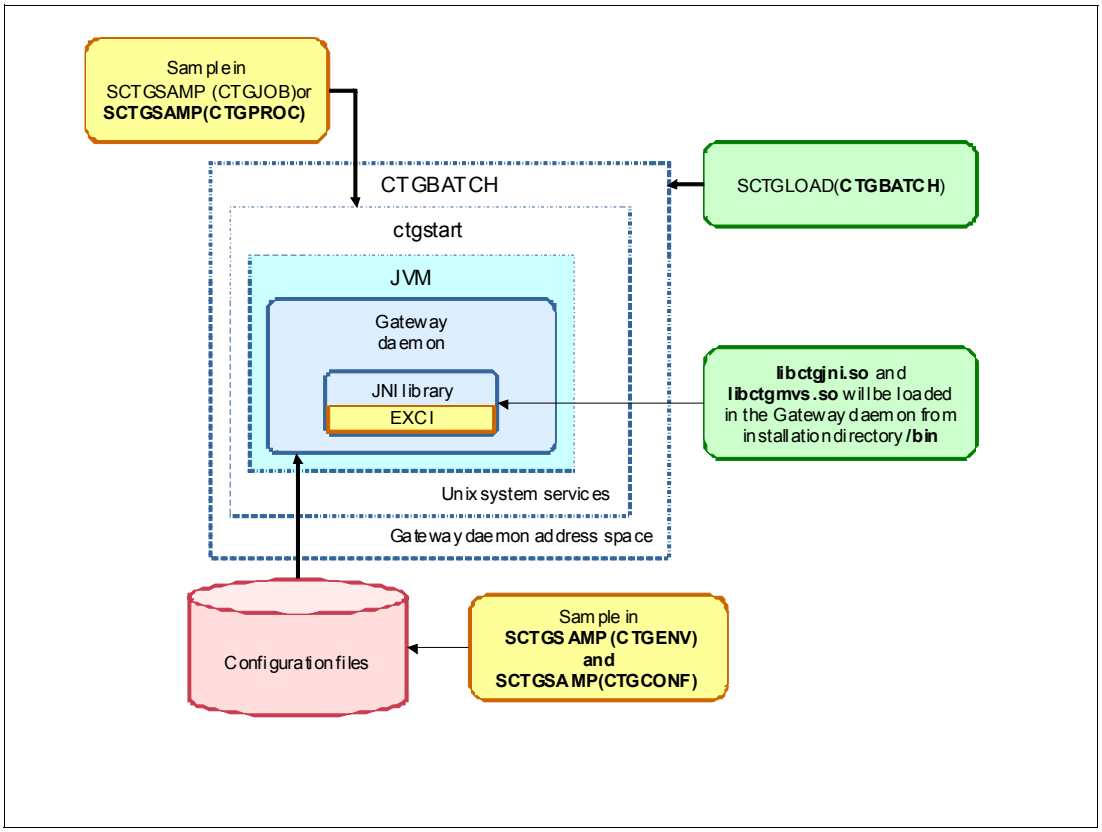


Figure 2-1 CICS TG runtime components

2.3.1 Gateway daemon

When implementing a 3-tier solution, at least one Gateway daemon needs to be started because it is the CICS TG component used to listen on a TCP/IP protocol port for client requests.

The Gateway daemon can be started as a procedure (a sample is supplied in member `CTGPROC` of `SCTGSAMP`), or as a batch job (a sample is supplied in member `CTGJOB`).

To benefit from the z/OS started task management, it is suggested that you launch the Gateway daemon as a procedure. Commonly, there are specific security policies related to user ID assignment to batch jobs. These policies can influence the need to have a privileged user assigned to the Gateway daemon. Another factor is the selection of the WLM Service Classes where the Gateway daemon has to run. Based on needs, the selection of service classes can provide a performance benefit. It can depend on how CICS TG address space is started.

However you start the Gateway daemon (procedure or batch job), the CTGBATCH is invoked.

CTGBATCH is an IBM Language Environment® batch mode program, which is supplied with CICS TG to start the Gateway daemon using a JCL started task. This program also passes the required environment variables to the UNIX System Services **`ctgstart`** script and routes messages to z/OS data sets and the JES log.

The **`ctgstart`** command will then create a Java virtual machine (JVM) where the Gateway daemon will run.

Gateway daemon behavior is influenced by the following configuration files:

- The *environment variables* can be specified in different ways (for example, in the `STDENV DD` statement, a z/OS data set, or even a z/OS UNIX file). For a detailed description, see the CICS TG Information Center:

  http://ibm.co/1rxRuFr

  A sample is provided in the data set member `CTGCONF` of `CTGSAMP` data set, or in <*install_path*>/`samples/configuration/ctgenvvarsamp`. The environment variables also contain the name of the configuration file that has to be loaded (**`CICSCLI`** parameter).

- The *configuration file* (also called the *ini file*) is a z/OS UNIX file or MVS data set that has to be provided to initialize the CICS TG. A sample is provided in member `CTGCONF` of `CTGSAMP` or in the <*install_path*>/`samples/configuration/ctgsamp.ini`. The configuration file, at a minimum, must contain the `GATEWAY` SECTION with a TCP protocol handler definition identifying the IP address and the port on which the Gateway daemon will listen.

  The default configuration used by CICS Transaction Gateway is the z/OS UNIX file:

  <*install_path*>/`bin/ctg.ini`

The Gateway daemon is a Java program that resources some native libraries, also called Java Native Interface (JNI) libraries, to interface with the z/OS system. They enable the Gateway daemon to interact with the z/OS console, to extract statistics information, such as the memory utilization, and to use sophisticated functions, such as Resource Recovery Services (RRS), IBM RACF® interface, and external CICS interface (EXCI) connections with CICS Transaction Server (CICS TS).

2.4 Adding facilities to CICS TG

This section describes the CICS TG dependencies on z/OS components or products for additional implementation, such as monitoring, security, restarting, and connectivity.

Figure 2-2 on page 27 shows how the z/OS elements interact with CICS TG.

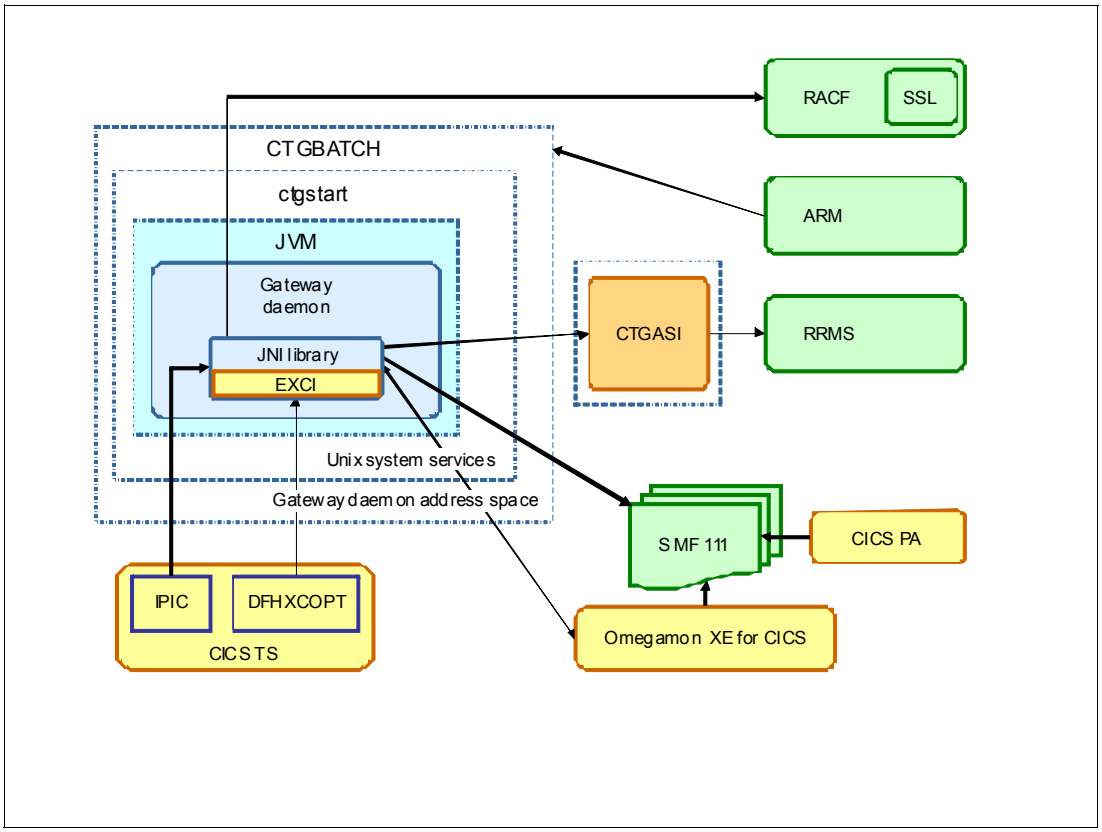


Figure 2-2 How CICS TG components interact with z/OS components

2.4.1 Automatic restart of a CICS TG region

You can benefit from the ability of z/OS to control the Gateway daemon automatic restart, by implementing the Automatic Restart Manager (ARM) facility. CICS TG can then use ARM for the automated restart of an MVS subsystem.

MVS automatic restart management is a sysplex-wide integrated automatic restart mechanism that can restart the following components:

- An abended MVS subsystem (or if a monitor program notifies ARM of a stall condition)
- A failed MVS image

Member `CTGPROC`, in the `SCTGSAMP` data set, provides a sample startup procedure to register and deregister to ARM using the `CTGARM` module.

Note: To use `CTGARM`, the `SCTGAUTH` load library must be defined as APF-authorized.

If you need detailed information about the ARM configuration, see *CICS Transaction Gateway for z/OS Version 6.1*, SG24-7161.

2.4.2 Using EXCI to communicate with CICS TS

The external CICS interface (EXCI) is an application programming interface (API). This API enables a non-CICS program running in z/OS to invoke a CICS program to pass and receive data.

This programming interface allows a user to allocate and open sessions (pipes) to a CICS system and to pass distributed program link (DPL) requests over them. CICS inter-region communication (IRC) supports these requests and each pipe maps onto one multiregion operation (MRO) session.

CICS TG acts as a non-CICS program and calls a remote program in CICS TS using DPL, passing `COMMAREA` data.

To set up the EXCI environment, CICS TG needs to load some CICS TS modules and the behavior can be controlled using the macro `DFHXCOPT`.

If you need additional information about the configuration of the EXCI connection to CICS, see *CICS Transaction Gateway for z/OS Version 6.1*, SG24-7161.

Note: The configuration of EXCI server connections is the same for both local and remote modes.

2.4.3 Clients passing a large amount of data to the CICS TS application

If your application needs to manage data larger than 32K, you can benefit from the implementation of the channel and containers solution.

To benefit from the channel and containers solution, you need to implement the IPIC connection to CICS TS. You can also benefit from the increased number of sessions, as compared to using the EXCI interface.

For more information about IPIC considerations, see 4.1.2, “IP interconnectivity (IPIC)” on page 45.

2.4.4 Client applications needing transactionality over EXCI

If the client application needs to maintain the data integrity over an EXCI connection, extended Logical Unit of Work (LUW) is used by the client.

To provide the infrastructure needed to guarantee the transactionality, CICS TG implements the use of unauthorized Resource Recovery Services (RRS). For a complete explanation of transactional models, see 6.3, “Transactional models” on page 97.

Note: Resource recovery consists of the protocols and program interfaces that allow an application program to make consistent changes to multiple protected resources.

Recoverable Resource Management Services (RRMS) consists of three z/OS components:

- Registration Service
- Context Services
- Resource Recovery Services

If you need additional information about the configuration of extended LUW support in CICS TG, see *CICS Transaction Gateway for z/OS Version 6.1*, SG24-7161.

2.4.5 Client applications needing distributed LUWs

If the client application needs to maintain the data integrity in different systems, the extended architecture (XA) protocol is needed and the XA support in the Gateway daemon must be enabled. For more information about this topic, see 6.3.1, “Two-phase commit” on page 98.

To provide support for XA transactions, the Gateway daemon makes use of RRMS facilities that require the calling address space to be registered as an authorized resource manager. A new address space called CTGRRMS acts as an agent for the Gateway daemon, providing access to the RRS capabilities required to support XA transactions.

CTGRRMS is a non-terminating address space, which will usually remain active until the next IPL of the z/OS image or logical partition (LPAR). CTGRRMS does not execute code itself beyond its own initialization. However, it provides services used by any Gateway daemon in the same LPAR that supports XA transactions.

During initialization, a Gateway daemon with XA transaction support enabled checks the availability of the CTGRRMS address space. If CTGRRMS is not available, the Gateway daemon requests the *CTGRRMS Address Space Initiator* (`ctgasi`) to start `CTGRRMS`.

Note: Member `CTGINIT` of `SCTGLINK` must be APF-authorized.

If XA is implemented over the EXCI protocol, all the information about the transaction will be stored in RRS.

If XA is implemented over IPIC connections, CICS TG needs to store only the CICS APPLID that received the request in Dynamic Server Selection (DSS). Failing XA transactions can be resolved only by the CICS region that started them. If DSS is used across LPARs, the XA coordinator cannot know which CICS received the request and then tries the recovery on the initial CICS.

If you need more information about the XA configuration, see *CICS Transaction Gateway for z/OS Version 6.1*, SG24-7161.

2.4.6 Using historical statistical data analysis

CICS TG for z/OS can take advantage of z/OS System Management Facility (SMF) to record statistical data in SMF type 111 records. The statistics collection can be interval or end-of-day driven. Members `CTGSMFRD`, `CTGSMFR`, and `CTGSMFB` in `SCTGSAMP` are provided to generate a user program to extract the SMF data, and run it.

SMF111 can be also analyzed by OMEGAMON XE for Messaging and the CICS Performance Analyzer tool. It is also possible to view the statistics data with CICS Explorer. See Chapter 14, “Explorer plug-in” on page 321.

If you need more information about monitoring and statistics, see *The Complete Guide to CICS Transaction Gateway: Volume 3 Systems Monitoring*, SG24-7562-01.

2.4.7 Using real-time statistical data analysis

CICS TG provides a statistical API, which is dynamically called during run time. Member `CTGSTATS` in the `SCTGDLL` library provides the CICS TG API.

IBM Tivoli OMEGAMON XE for CICS on z/OS implements this functionality to provide real-time statistics data.

For more information about monitoring and statistics, see *The Complete Guide to CICS Transaction Gateway: Volume 3 Systems Monitoring*, SG24-7562-01.

2.4.8 Multiple Gateway daemons started on the same LPAR

If more than one CICS TG is started in the same LPAR, there is redundancy on the modules loaded. For this reason, you can benefit from creating a Java shared class cache. It allows multiple Gateway daemons, within an LPAR, to consolidate class storage. The Java shared class cache is automatically created by default and it is called `cicstg`*`xxx`*, where *xxx* is the CICS TG version, modification, and level. It can be disabled in **`ctgstart`**, by coding the option `-nosharedclasses`.

2.4.9 Security

When connecting client applications to CICS, it is important to maintain appropriate controls over CICS programs, data, and other resources. With the CICS TG, the infrastructure and APIs enable authentication of the client system and the user. After the user has been authenticated, CICS, in conjunction with the external security manager, can apply the authorization rule to restrict access to sensitive applications and data.

If the connection between client/server applications needs to preserve confidentiality for the data being transmitted over the network, CICS TG can benefit from support for the standard SSL and Transport Layer Security (TLS) protocols to encrypt the transmitted data.

For a more thorough discussion of the security considerations, mechanisms used to secure connections, and transmission of security credentials to the CICS system, see Chapter 8, “Security” on page 131.

2.4.10 Setting up high availability

CICS TG can use different strategies to guarantee the quality of service required, such as Dynamic Server Selection (DSS) and CICS TG Gateway group technologies. These technologies can help you in implementing a high availability system even when your business transactions create some affinities between specific entities.

Various solutions are described in more detail in Chapter 7, “High availability concepts” on page 103.

For an example and more information, see *Using CICS Transaction Gateway, High Availability, and the CICS Explorer*, REDP-4782.

2.5 The benefits of CICS TG for z/OS

CICS Transaction Gateway is available on a variety of platforms, providing different capabilities and qualities of service.

Each product delivers the core capability for enterprise modernization and connectivity for CICS assets. However, depending on deployment choice, CICS TG can deliver different benefits.

For a detailed table and an overview of the broad benefits achievable with different CICS TG products, visit the following link:

http://www.ibm.com/software/products/us/en/cics-ctg/

3

CICS Transaction Gateway for Multiplatforms and Desktop Edition installations

This chapter describes the installation and file structure of CICS Transaction Gateway (CICS TG) on both the Windows and AIX platforms. This includes the resulting file structure of installed components and the location of key components. This section contains details for both Multiplatforms and Desktop Edition.

The following concepts are covered in this chapter:

3.1 Preparing to install CICS TG Multiplatforms and CICS TG Desktop Edition in a distributed environment

The CICS TG installation image can be obtained from either IBM or an authorized IBM Business Partner. These sources can be accessed through the IBM Passport Advantage® site.

3.1.1 The Passport Advantage site

The CICS TG is made available through an installable image referred to as the *eImage*. This is true regardless of the platform, which includes AIX, HPUXIA64, Linux (Linux on Intel, Linux on Power, and Linux on System z), Solaris, and Windows.

The Passport Advantage site is a single point of contact to download and manage your purchased software. To use this site, valid IBM identification is required. You are limited to software for which you hold a valid license.

Passport Advantage Online enables you to perform the following actions:

- Access entitled software: Download or request a media pack.
- Renew IBM software subscription and support or acquire licenses for your sites by browsing the popular offerings catalog or IBM quotes (available in many but not all countries).
- Review and update contact information for your site's designated contacts.
- View Proof of Entitlement certificates for your sites.
- Subscribe to eNotifications to receive automatic notification when new software upgrades, matching your preferences and entitlement, become available.

  Note: You must be authorized by your site's primary contact or primary contact designate to take advantage of eNotifications.

- Generate reports to track transaction history, download history, or active entitlements at the enterprise or site level.

The Passport Advantage site can be accessed by the following link:

`http://www.ibm.com/software/lotus/passportadvantage/`

3.1.2 Files provided within the installable eImage

For the following example, the Windows platform will be used. The contents of the downloaded eImage are shown in Figure 3-1 on page 35.

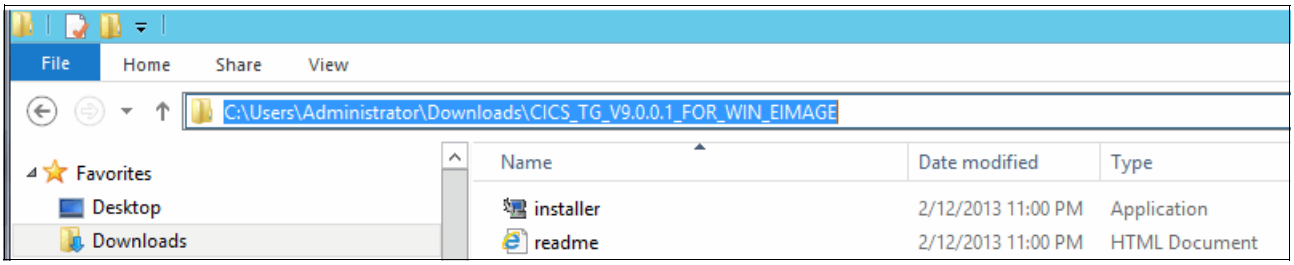


Figure 3-1 eImage contents

The following files are contained in the download and described in the following list:

- The `installer.exe` executable file contains the installation media and the installer itself.
- The `readme.html` file contains installation information, such as installation instructions, upgrade instructions, and any service updates that are included.

3.2 Anatomy of CICS TG in a Windows environment

The file structure of an installed CICS TG on Windows is illustrated in Table 3-1.

Table 3-1 Installed file contents

Directory name	Purpose
CICS Transaction Gateway	Product root directory, install response file, and `product.xml` version file
bin	Executable binaries
classes	CICS TG Java classes
copybook	Development COBOL copybooks
deployable	Deployable EAR files, RAR files, and `ctgredist.zip` containing the redistributable components for developing C and .NET applications on AIX, HPUXIA64, Linux on IBM System i®, Linux on IBM Power Systems™, Linux on System z, Solaris Sparc, and Windows
docs/javadoc	API documentation reference for Java, CICS TG ECI and ESI Version 2, .NET, and CICS TG Statistics
include	Development C and C++ header files
jvm17	Java software development kit (SDK)
lib	Product lib and dll files
lib64	64-bit `ctgclient.dll`
license	Product license text
msgs	Product message text and help files
properties	CICS TG product version files
samples	CICS TG samples and scenarios files
uninstall90	uninstall executable

The CICS TG is designed to run as a Windows service. It is started and stopped through this service, as illustrated in Figure 3-2.

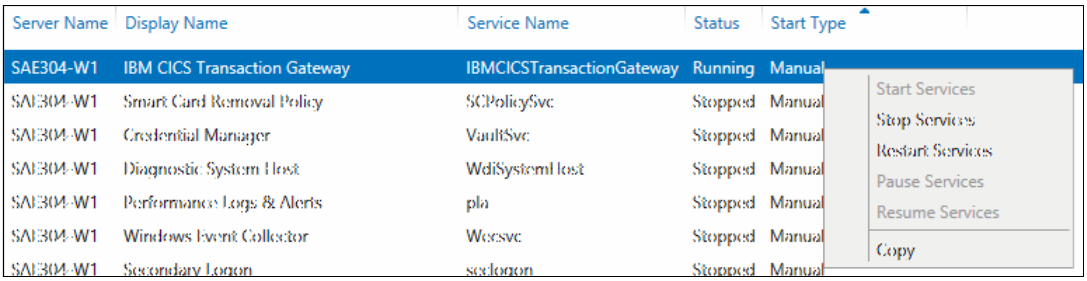


Figure 3-2 Running CICS TG Windows service

Now that the CICS TG is started, as shown in Figure 3-2, the running processes for the CICSTG (daemon and service) can be verified as running as shown in Figure 3-3.

Figure 3-3 Active processes of a running CICS TG on Windows

3.3 Anatomy of CICS TG on UNIX

For the following example, the AIX platform will be used. The contents of the downloaded eImage are shown in Figure 3-4.

```
s1(root)/tmp/ctg_9_install_image:>gunzip -c ctg_9_install.tar.gz | tar -xvf-
x installer, 155880741 bytes, 304455 tape blocks
x readme.txt, 14937 bytes, 30 tape blocks
s1(root)/tmp/ctg_9_install_image:>ls -la
total 605600
drwxr-xr-x    2 root     system          256 04 Apr 16:51 .
drwxrwxrwt   28 bin      bin           20480 04 Apr 16:50 ..
-rwxr-xr-x    1 root     system    154136615 04 Apr 16:41 ctg_9_install.tar.gz
-rwxr-xr-x    1 11889    11889     155880741 15 Nov 00:00 installer
-rw-r--r--    1 11889    11889         14937 15 Nov 00:00 readme.txt
s1(root)/tmp/ctg_9_install_image:>
```

Figure 3-4 Unpacking the CICS TG for Multiplatform installation image contents on AIX

The following files are contained in the download and described in the following list:

- The `installer` executable file contains the installation media and the installer itself.
- The `readme.txt` file contains install information, such as instructions for installing and upgrading, and any service updates that are included.

The file structure of an installed CICS TG on AIX is illustrated in Table 3-2.

Table 3-2 Installed file contents

Directory name	Purpose
CICS Transaction Gateway	Product root directory, install response file, and `product.xml` version file
bin	Executable binaries
classes	CICS TG Java classes
copybook	Development COBOL copybooks
deployable	Deployable EAR, RAR files, and `ctgredist.zip` containing the redistributable components for developing C and .NET applications on AIX, HPUXIA64, Linux on System i, Linux on Power Systems, Linux on System z, Solaris Sparc, and Windows
docs/javadoc	API documentation reference for Java, CICS TG ECI and ESI Version 2, .NET, and CICS TG Statistics
icons	gui icon `.gif` files
include	Development C and C++ header files
installlogs	Current installation log files
jvm17	Java SDK
lib	Product lib and dll files
license	Product license text
msgs	Product message text and help files
properties	CICS TG product version files
samples	CICS TG samples and scenarios files
uninstall90	uninstall executable

The CICS TG on AIX runs as a daemon and is started and stopped using the **`ctgd`** daemon script. This not only starts and stops the CICS TG, but it also allows it to run as a background process. For more information about configuring and using the CTGD, see Chapter 11, “CICS Transaction Gateway Multiplatforms and Desktop Edition configuration” on page 205.

3.4 The CICS TG and Client daemons and runtime components

Regardless of the distributed platform on which the CICS TG is installed, all versions share the same components. Figure 3-5 illustrates the components of CICS TG for Multiplatforms and its positioning in the product architecture. The JCA resource adapters are shipped as part of the CICS TG product for deployment to a supported Java EE application server (for example, WebSphere Application Server).

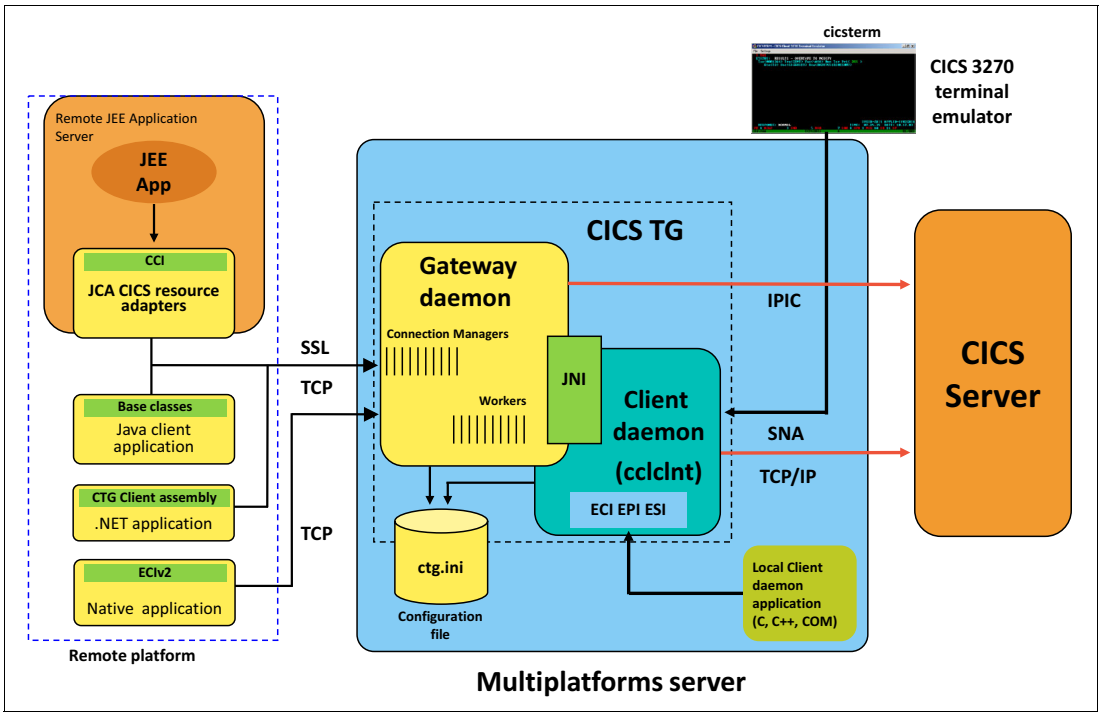


Figure 3-5 Remote mode components of CICS TG for Multiplatforms product

3.4.1 The Client daemon

The Client daemon is an integral part of CICS TG on all distributed platforms. The Client daemon provides the local interface to C, C++, COM, and COBOL client applications to enable communication with CICS. It provides Systems Network Architecture (SNA) and TCP/IP protocol connectivity to CICS using the same technology as IBM CICS Universal Client. You can configure these protocols with other protocol-specific parameter values using the CICS TG Configuration tool (**`ctgcfg`**) or by directly editing the CICS configuration file.

3.4.2 The Gateway daemon

The Gateway daemon listens for incoming remote client requests and manages the threads and connections necessary to ensure good performance. The Gateway daemon handles remote client requests using its protocol handlers component. The Gateway daemon can be configured to support both TCP/IP and Secure Sockets Layer (SSL) protocols. You can configure these protocols and set Gateway daemon-specific parameters using the CICS TG Configuration tool (**`ctgcfg`**).

3.4.3 The IPIC protocol

The CICS Transaction Server for z/OS Version 3.2 (CICS TS) introduced the IPIC protocol. It is used for intercommunication with CICS regions over TCP/IP. CICS TG V7.1 or later supports the IPIC protocol. The IPIC support is provided in the CICS TG as a separate component called the *IPIC protocol driver*. This driver is independent of the Client daemon component. The IPIC configuration for local mode is different from local mode configuration supported by other protocols with the Client daemon. The IPIC protocol can be configured by using the CICS TG Configuration tool (**ctgcfg**) or by directly editing the CICS configuration file. More information about local and remote mode configuration is available in Chapter 6, "CICS Transaction Gateway solution options" on page 89.

The following limitations apply when connecting using the IPIC protocol:

- You cannot use the EPI or ESI application programming interfaces.
- You cannot use the ECI asynchronous calls.
- You cannot use the **cicscli** command for CICS server definitions.

3.4.4 The configuration tool

Configuration settings are stored by default in the `ctg.ini` file, the `<install_path>/bin` directory on UNIX and Linux, and in the `<product_data_path>` directory on Windows. You can specify a different location and optionally change the name of the configuration file by setting the **CICSCLI** environment variable. A sample configuration file (`ctgsamp.ini`) is in the `C:\Program Files (x86)\IBM\CICS Transaction Gateway\samples\configuration` for Windows and `/opt/IBM/cicstg/samples/configuration` for UNIX.

> **Note:** Before the release of Version 7.2 of CICS TG, the `ctg.ini` was by default in the product `bin` directory on Windows.

The **ctgcfg** utility can be used for the following activities:

- CICS TG APPLID, APPLID Qualifier, Default Server, key ring file, and key ring password can be configured on the CICS Transaction Gateway page.
- In the Gateway daemon section, you can configure gateway-supported protocols: TCP/IP and SSL. You can also configure Gateway daemon-specific resources, logging, monitoring facilities, and Statistics API settings.
- In the Client daemon section, you can configure the Client daemon-specific resource and the logging facilities. You can also configure the Workload Manager (WLM)-specific parameters. WLM is supported only on the Windows platform.
- In the CICS Servers section, you can add new server connection definitions to CICS by right-clicking the CICS servers in the navigation panel of the **ctgcfg** tool. You can select TCP/IP, SNA, IPIC, and named pipes (supported only on the Windows platform) protocols to connect to CICS.

  > **Note:** Support for named pipes was removed in CICS TG Version 8.1.

More information about using the **ctgcfg** utility is provided in Chapter 11, "CICS Transaction Gateway Multiplatforms and Desktop Edition configuration" on page 205.

3.4.5 Resource adapters

Resource adapters are used in support of Java EE Connector architecture client applications. To understand resource adapters, it is important to understand Java EE Connector architecture.

Purpose of Java EE Connector architecture

Java EE Connector architecture defines a standard architecture for connecting the Java Platform Enterprise Edition (Java EE) platform to heterogeneous enterprise information systems (EIS). Examples of an EIS include transaction processing systems, such as the CICS TS and Enterprise Resource Planning systems (SAP).

The connector architecture enables an EIS vendor to provide a standard resource adapter. A resource adapter is a middle-tier between a Java application and an EIS, which enables the Java application to connect to the EIS. A resource adapter plugs into application servers supporting the Java EE Connector architecture.

Part 2

CICS Transaction Gateway capabilities

Part 2 shows you how to choose the appropriate CICS connection protocol, APIs for the applications, and security options and understand the services available to the application developer through that protocol. You also look at 3-tier solutions and those CICS Transaction Gateway (CICS TG) capabilities that provide high availability.

4

Connecting to CICS

This chapter describes the communication protocols available for connecting the CICS Transaction Gateway (CICS TG) products with CICS server products. It describes at a high level the APIs supported by each protocol and the qualities of service provided.

This chapter contains the following topics:

4.1 CICS communication protocols

This section provides an overview of the various communication protocols available for use with CICS Transaction Gateway products and supported CICS server products. A more detailed technical analysis of their capabilities and qualities of service is provided in 4.3, “Quality of service and connection types” on page 50.

Over time and advances in technology, a mixture of wire protocols has been implemented by the CICS Family of products, each with its own characteristics and capabilities. There are now many possible combinations of products and protocols. It is often difficult to decide which will satisfy a specific set of application requirements.

Table 4-1 summarizes the communication protocols available to connect CICS TG products with CICS Transaction Server (CICS TS) and TXSeries products.

Table 4-1 Capabilities for CICS TG and CICS server products

CICS protocol	Available with CICS TG products	Connects with CICS server products
IPIC	z/OS Multiplatforms Desktop Edition	CICS TS for z/OS TXSeries for Multiplatforms
External CICS interface (EXCI)	z/OS	CICS TS for z/OS
TCPIP	Multiplatforms Desktop Edition	CICS TS for z/OS CICS TS for z/VSE TXSeries for Multiplatforms CICS TS for System i
SNA	Multiplatforms Desktop Edition	CICS TS for z/OS CICS TS for z/VSE CICS TS for System i

4.1.1 What’s in a name?

Before proceeding any further, it is important to provide historical information regarding the names of the CICS connection protocols, known as *TCPIP*, *ECI over TCP/IP*, or *ECI over IP*.

Note: The CICS TCPIP protocol (without a slash) is a CICS Family communications protocol, operating over TCP/IP networks (with a slash). However, it is not the only CICS communications protocol to operate over TCP/IP networks.

At the time of its inception, the world of external CICS communications was viewed as being either Systems Network Architecture (SNA) or TCP/IP-based. The protocol drivers in the CICS TG products became known as SNA and TCPIP. Although convenient at the time, the choice of TCPIP subsequently proved to be too generic, and today causes confusion due to other CICS protocols also being built on top of TCP/IP networks. Today, both the IPIC and TCPIP CICS protocols operate over a TCP/IP network, as well as HTTP/HTTPS, Web Services, and no doubt more to follow in the future.

The TCPIP CICS Family protocol is often referred to as *ECI over TCP/IP* in the context of CICS TS for z/OS, or CICS TS for z/VSE, because those particular CICS server products only support ECI requests when using this protocol. Also, it is defined as “Protocol: ECI” in a CICS TCPIPService resource definition. However, the TXSeries and CICS TS for System i also support external presentation interface (EPI) requests with this protocol, and so ECI over TCP/IP is not always an appropriate name for this CICS Family protocol.

The subsequent sections of this topic provide a summary of each CICS communication protocol, in the context of the CICS TG products, CICS server products, and the CICS TG APIs.

4.1.2 IP interconnectivity (IPIC)

IPIC is a type of intersystem communications, introduced in CICS TS for z/OS V3.2. It provides a TCP/IP network-based alternative to intersystem communication (ISC) over Systems Network Architecture (ISC over SNA). It supersedes the older TCPIP protocol for ECI applications, providing enhanced capabilities, qualities of service, and cross-platform availability.

IPIC connections enable the integration of CICS-to-CICS, and CICS TG-to-CICS communications, into a TCP/IP-based network infrastructure. It also enables the use of Secure Sockets Layer (SSL) to provide secure connectivity.

IPIC is the only CICS protocol implemented by all of the CICS TG products, and is therefore available on all supported platforms. Before the CICS TG V7.1 products delivered support for IPIC, the only connectivity available with the CICS TG for z/OS product was EXCI.

IPIC provides ECI access to CICS TS for z/OS and TXSeries products over the TCP/IP network, supporting both COMMAREA and channel programs, and two-phase commit. Additionally, IPIC connections to CICS TS for z/OS support the external security interface (ESI) for password verification and change. IPIC does not provide external presentation interface (EPI) support.

Figure 4-1 shows how IPIC connectivity can be used with CICS TS for z/OS V3.2, or later. This general pattern is true for all CICS TG deployments, except that JCA resource adapters are not included with CICS TG Desktop Edition.

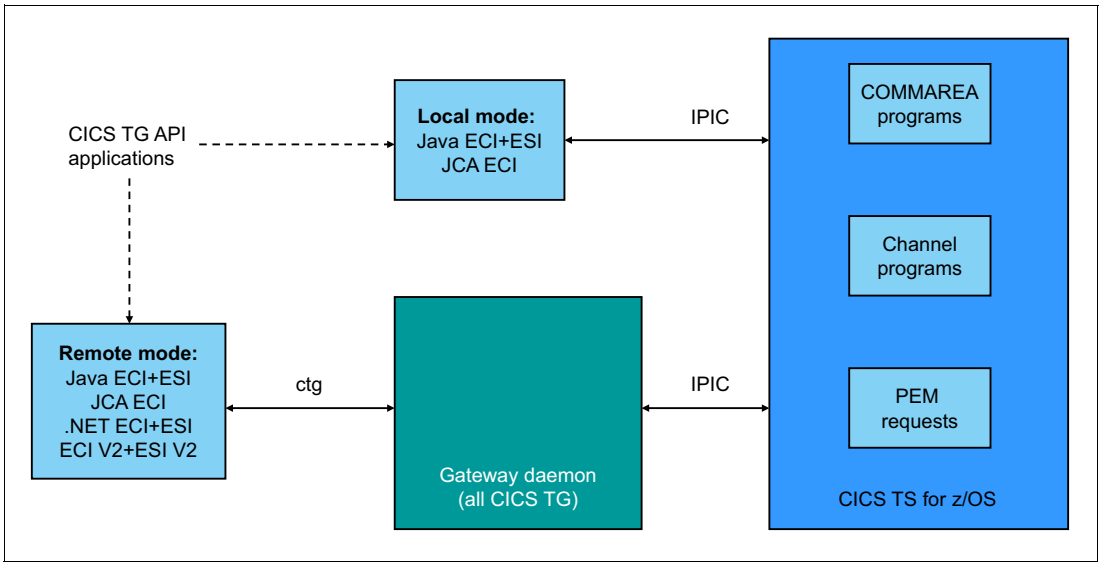


Figure 4-1 IPIC connectivity with CICS TS for z/OS

Figure 4-2 shows how IPIC connectivity can be used with TXSeries for Multiplatforms V7.1, or later. This general pattern is true for all CICS TG deployments, except that the JCA resource adapters are not included with CICS TG Desktop Edition.

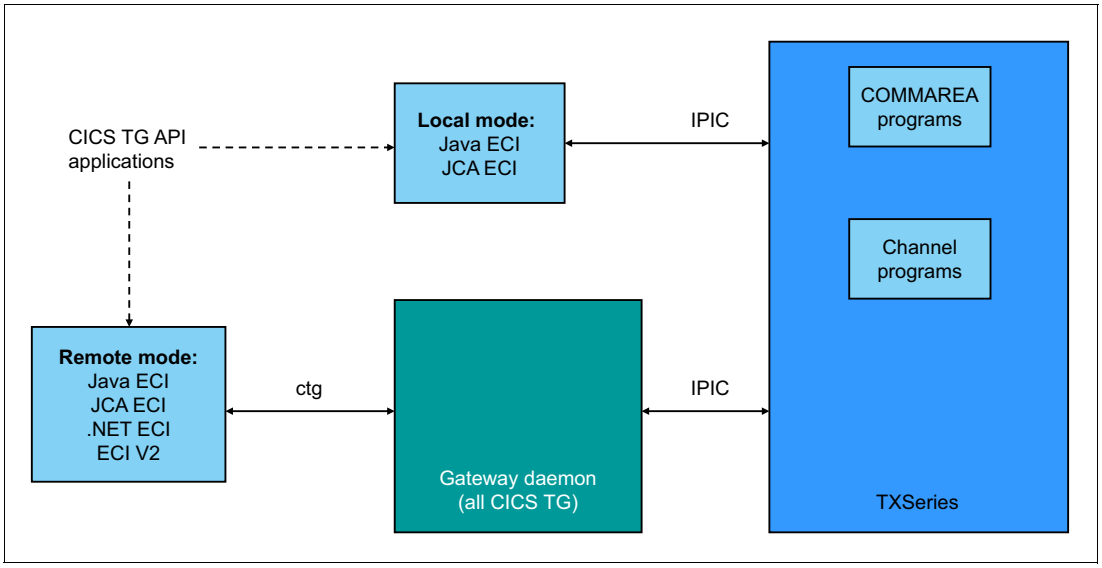


Figure 4-2 IPIC connectivity with TXSeries

Note: The high-level difference between CICS TS for z/OS and TXSeries in terms of support for IPIC connectivity is that TXSeries does not support external security interface (ESI) requests for password expiry management, and it does not support password phrases.

4.1.3 EXCI

The External CICS Interface (EXCI) provides a means for non-CICS address spaces on z/OS to link to programs in a CICS region. For example in Figure 4-3 on page 47, EXCI allows batch jobs to connect to CICS and also allows the CICS Transaction Gateway for z/OS to make calls to CICS. The use of EXCI imposes some limits on both the client program and the CICS server program. The invoked CICS program must be a COMMAREA-based program. For the client application, use of EXCI limits the ECI request to only that portion which EXCI supports, and prevents use of external presentation interface (EPI) to invoke 3270 applications or ESI to invoke password functions in CICS.

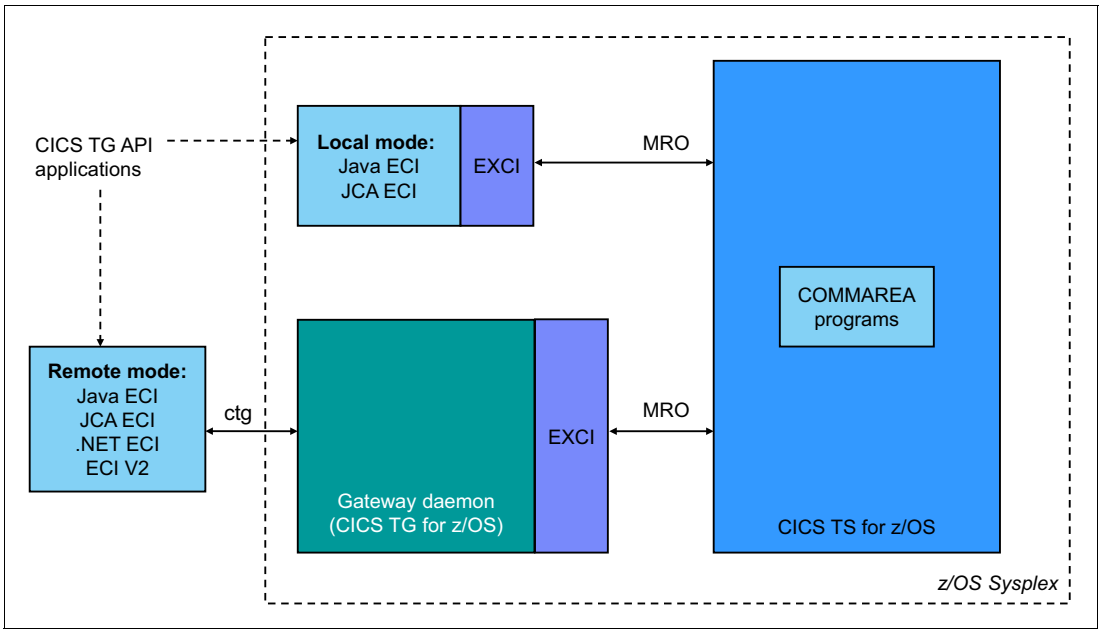


Figure 4-3 EXCI connections with CICS TS for z/OS

4.1.4 TCPIP

When connecting to TXSeries and CICS TS for System i servers, the TCPIP connection protocol supports both ECI and EPI application programming interfaces. However, ECI requests are limited to COMMAREA-based application requests. When connecting to CICS TS for z/OS and CICS TS for VSE servers, this protocol supports only COMMAREA-based ECI requests, and is also referred to as ECI over TCP/IP.

Note: The TCPIP connection protocol is not supported with the CICS TG for z/OS product.

Figure 4-4 on page 48 shows how TCPIP connectivity can be used with CICS TS for z/OS and CICS TS for VSE. This general pattern is true for all CICS TG for Multiplatforms and CICS TG Desktop Edition deployments, except that JCA is not included with CICS TG Desktop Edition.

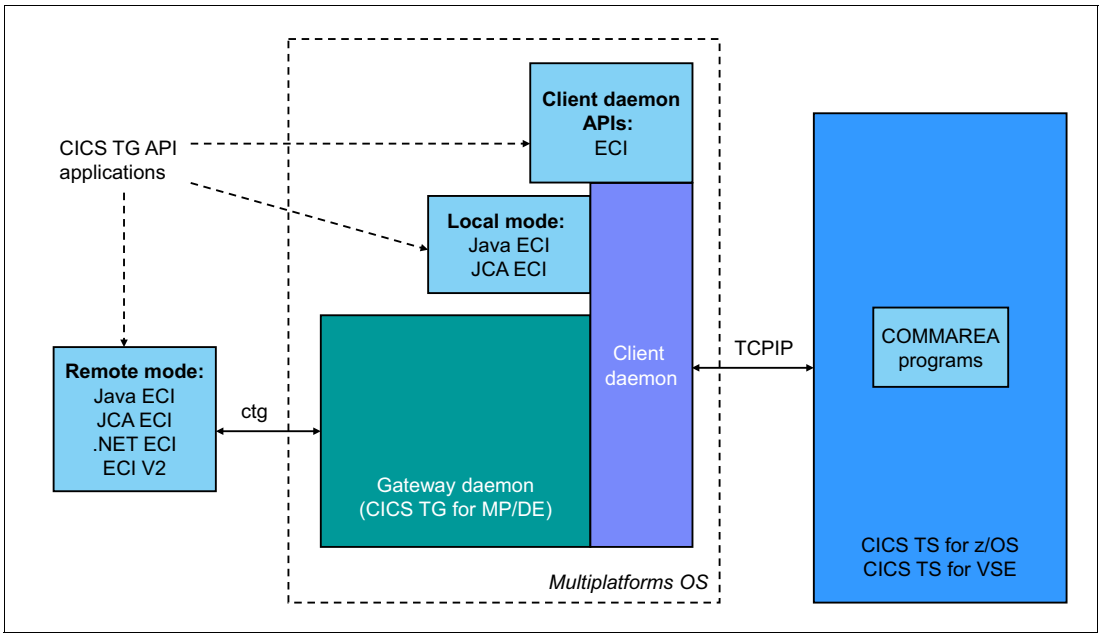


Figure 4-4 TCPIP connections with CICS TS for z/OS and CICS TS for z/VSE

Figure 4-5 shows how TCPIP connectivity can be used with TXSeries and CICS TS for i. This general pattern is true for all CICS TG for Multiplatforms and CICS TG Desktop Edition deployments, except that JCA is not included with CICS TG Desktop Edition.

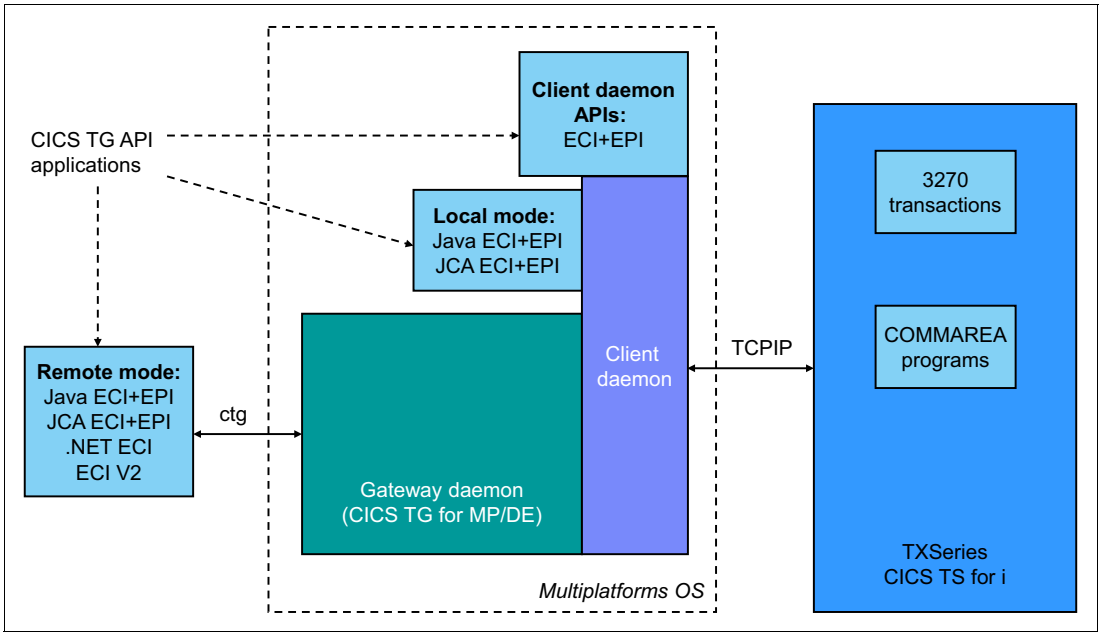


Figure 4-5 TCPIP connections with TXSeries and CICS TS for i

4.1.5 SNA Advanced Program-to-Program Communication (APPC)

Advanced Program-to-Program Communication (APPC), also known as Logical Unit 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. SNA connections are only supported in CICS TG for Multiplatforms and CICS TG Desktop Edition. As shown in Figure 4-6, the SNA connection protocol supports all of the gateway APIs to all CICS servers except TXSeries.

Note: An additional product, such as IBM Communication Manager, is required to provide SNA connectivity.

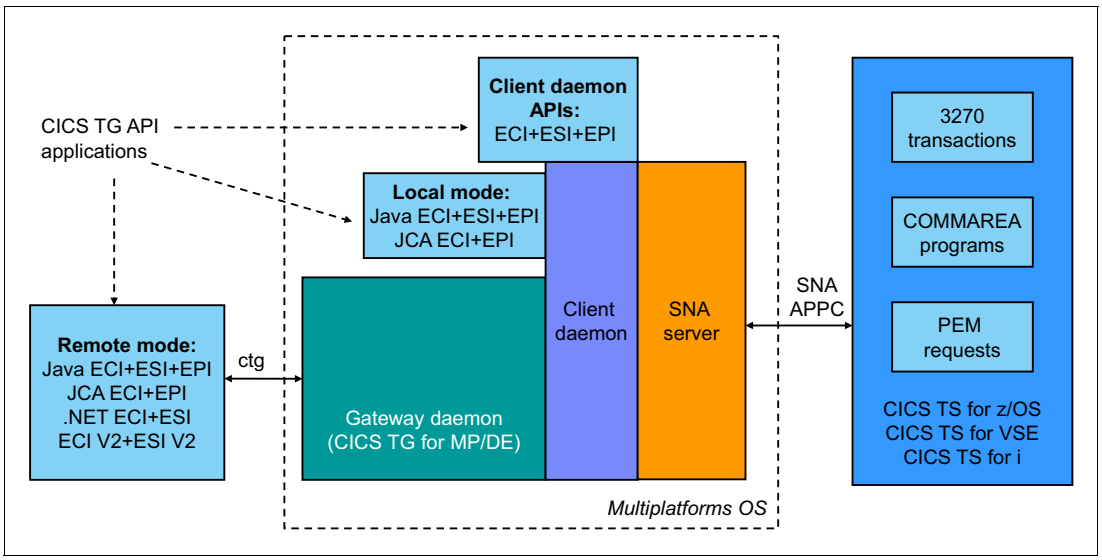


Figure 4-6 SNA APPC connections with the CICS TS family of products

4.2 CICS TG APIs and connection type

All CICS TG products provide both ECI and ESI programming interfaces for multiple programming languages. ECI provides access to CICS programs, with data encapsulated as COMMAREAs or channels and containers. ESI provides access to Password Expiry Management (PEM) capabilities, such as verify password or change password.

The CICS TG for Multiplatforms and CICS TG Desktop Edition products additionally provide the EPI for access to CICS terminal transactions through virtual 3270 sessions.

Note: Not all of the CICS TG programming interfaces are available in each CICS TG product, and not all programming interfaces included within a CICS TG product are available for use with all types of CICS connection. The choice of programming API might impose limitations on the connection type that you use.

Table 1-1 on page 14 summarizes the capabilities and options for connectivity between combinations of CICS TG and CICS server products. Table 4-2 on page 50 presents the same information, from the viewpoint of which API is required.

Table 4-2 Options for connectivity between combinations of CICS TG products and APIs

API	IBM CICS connection		
	CICS TG for z/OS	**CICS TG for Multiplatforms**	**CICS TG Desktop Edition**
ECI [a]	EXCI IPIC	IPIC SNA TCPIP	IPIC SNA TCPIP
ESI[b]	IPIC	IPIC SNA	IPIC SNA
EPI	N/A	SNA TCPIP [c]	SNA TCPIP [d]

a. Channels and containers are only available for ECI requests using IPIC CICS connections.
b. ESI is only applicable to IPIC connections to CICS TS for z/OS, or when using the SNA connection protocol to CICS TS for VSE and CICS TS for i.
c. EPI requests using TCPIP CICS connections are possible with TXSeries and CICS TS for i servers.
d. EPI requests using TCPIP CICS connections are possible with TXSeries and CICS TS for i servers.

The CICS TG information centers each include the topic, "*Which API can be used?*", which provides a full cross-reference between programming language API support and CICS communications protocol available for the CICS TG for Multiplatforms and CICS TG Desktop Edition products:

http://ibm.co/1n2ltV6

The following topic is equivalent for the CICS TG for z/OS product:

http://ibm.co/WnWuSN

4.3 Quality of service and connection types

This section takes each of the available CICS connection protocols and breaks out the capabilities and qualities of service.

There is a choice of network protocols for connecting to CICS. All protocols support ECI COMMAREA requests. Table 1-1 on page 14 shows the supported options for connecting to different CICS servers using the various protocols and client APIs. With the availability of IPIC connectivity and especially support for SSL on IPIC connections, it is now feasible to migrate many CICS TG solutions to IPIC, except for EPI-based applications. These will still require an SNA connection to support access to terminal-based transactions with CICS TS for z/OS and CICS TS for z/VSE. Because IPIC connections are the strategic direction for the CICS TG, this book will use IPIC connections in all examples and scenarios.

Note: EPI client applications connect to TXSeries using the TCPIP protocol rather than SNA, and to CICS TS for i using either TCPIP or SNA.

Performance is often a key factor in the evaluation of technology, but it is beyond the scope of this book. However, a good comparison of connection protocols is available in the IBM Redpaper™ publication, *IBM CICS Performance Series: A Processor Usage Study of Ways into CICS*, REDP-4906:

http://www.redbooks.ibm.com/abstracts/redp4906.html

4.3.1 IPIC

This topic describes characteristics of CICS connections using the IPIC protocol, from the perspective of the CICS Transaction Gateway.

Key differentiators

IPIC connectivity continues to evolve and mature since its introduction in 2007, and now offers multiple benefits over the other CICS Family communication protocols:

- Channels and container support for ECI provides an improved method of exchanging data with CICS programs in amounts that far exceed the 32-KB limit that applies to COMMAREAs and additionally provides an optimized and more structured data interface.
- Very high offload to specialty processors, such as IBM zSeries Application Assist Processor (zAAP), when using CICS TG for z/OS.

  Although the Gateway daemon is a Java-based address space, it switches to native code for all CICS communications protocols, except IPIC. When CICS TG for z/OS is deployed in a system with zAAP engines enabled, the offload will significantly reduce the general CPU cost per transaction.
- Higher concurrent workload capacity, especially compared to EXCI.
- Secure SSL/Transport Layer Security (TLS) connectivity to CICS TS for z/OS, or TXSeries, regardless of where the CICS TG product is deployed.
- Password phrases are supported for ECI and ESI with CICS TS for z/OS.
- Best transaction tracking capabilities, due to the provision of origin data, and integration with the Cross Component Trace (XCT) facility in WebSphere Application Server V8.5, or later.
- IPIC local mode is the only configuration offering extended architecture (XA) transaction support, for the CICS TG for Multiplatforms product.
- No support for EPI applications.

Because the CICS TG implementation of IPIC is Java and TCP/IP-based, it transparently benefits from ongoing improvements to these parts of the underlying software, for example:

- The IBM Java software development kit (SDK) regularly exploits the latest hardware offerings, for improvements in performance
- The IBM Java Secure Socket Extension (JSSE) library offers continually improving security capability, such as TLS 1.2 support, and exploitation of encryption hardware
- TCP/IP stack optimizations, for example, dynamic right sizing, or Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE) exploitation

Transactions

IPIC connections can be used for synconreturn, extended logical unit of work (LUW), and XA transaction-based ECI requests. While all of the CICS server products support synconreturn and extended LUW transactions, CICS TS for z/OS is the only CICS server product providing support for XA transactions with IPIC connectivity.

The CICS TG for z/OS product supports XA transactions when the ECI resource adapter is deployed in local mode, for example, with WebSphere Application Server for z/OS, or remote mode, for example, with a certified Java EE application server, on any platform.

The CICS TG for Multiplatforms product supports XA transactions when the ECI resource adapter is deployed in local mode. It does not support XA transactions in remote mode.

The CICS TG Desktop Edition does not support XA transactions, because the Java EE resource adapters are not included.

Note: Only the CICS TG for z/OS product provides support for XA transactions in a 3-tier (remote mode) topology. Therefore, only CICS TG for z/OS can provide the combination of XA transaction support and Dynamic Server Selection capabilities for high availability, such as DSSPOLICY, or the CICS Request Exit.

From the application programming perspective, XA transactions are available exclusively through the ECI resource adapter, when a connection factory is configured to enlist in a global transaction. This can be achieved with either container-managed transaction (CMT) or bean-managed transactions (BMTs).

Security

IPIC connectivity offers the greatest selection of security options within the range of the CICS Family communication protocols. It allows for the selective implementation of various types of security, from identity assertion through to SSL client authentication, with full control over cipher suites.

Bind time security can be used to apply a security check when a request to establish a connection is received from, or sent to, a remote system. Its purpose is to prevent an unauthorized system from connecting to CICS. For IPIC, bind security is supported by the exchange of Secure Sockets Layer (SSL) certificates.

Link security restricts the resources that a user can access, depending on the remote system from which they are accessed. The practical effect of link security is to prevent a remote user from attaching a transaction or accessing a resource for which the link user ID has no authority.

With CICS TS for z/OS, the authority of the link comes from one of these user IDs:

- A user ID that is specified on an `IPConn` definition
- A user ID that is associated with a certificate received from the partner system, through an External Security Manager

User security can be used to check the received security context that flows from the requestor and determine the user ID under which the transaction executes. For ECI requests, the received security context (user ID and either password, or password phrase) is usually specified by the ECI application, or for Java EE applications, it can be specified in the ECI connection factory custom properties: `userName` and `Password`. With IPIC connections using SSL, the received security context can be derived from a client certificate.

Identity assertion is a special form of user security, where the transaction operates under the user ID specified by the requestor, without a password or password phrase credential. This is allowed in CICS TS for z/OS with IPIC connections, when the partner system meets these conditions:

- In a z/OS logical partition (LPAR) within the same sysplex
- Using a secure SSL IPIC connection

Identity propagation can be viewed as an extension of identity assertion, allowing a pre-validated distributed identity, from a non-System Authorization Facility (SAF)-based security domain, to be propagated from WebSphere Application Server to CICS. Identity propagation is only available with IPIC connections.

For further detail about identity propagation support in CICS Transaction Gateway, see the Information Center topic, *Identity propagation*:

http://ibm.co/1mYB2xG

For full details, see *z/OS Identity Propagation*, SG24-7850:

http://www.redbooks.ibm.com/abstracts/sg247850.html

IPIC supports ESI requests, allowing application programs to perform password expiry management (PEM) operations, including change and verify password.

For full details about IPIC security with CICS TS for z/OS, see the Information Center topic, *Implementing IPIC security*:

http://ibm.co/WjtHyu

Capacity limits

Capacity for IPIC connections is determined by the number of sessions made available. The total number of IPIC sessions, across multiple connections, that can be created, are subject to available Java heap storage in either the Gateway daemon, or the Java EE application server (local mode).

Each IPIC connection is configured to have a specific number of logical sessions available, and a hard limit on that number is imposed by architectural limits in the CICS server product. Since the introduction of IPIC with CICS TS for z/OS V3.2, this limit remains at 999 sessions.

Given that multiple IPIC connections can be used concurrently, to multiple CICS server instances (or to a single CICS server instance), it is clear that the number of concurrent requests is now constrained only by the capacity for work in the connected CICS servers, the Gateway daemon (number of work threads), or the source of the workload, for example, WebSphere Application Server.

Note: With the CICS TG for z/OS product, full exploitation of IPIC capacity is only possible by disabling EXCI support. If EXCI support is not disabled, the maximum number of concurrent requests per Gateway daemon is constrained by the LPAR configuration value, `LOGONLIM`. This optional CICS configuration value has a default of 100 and a maximum possible value of 250.

At run time, a negotiation is made during connection establishment and the session limit is determined to be the lower of the two values, defined on each side.

Note: Before CICS TG V9.0, local mode IPIC connections always negotiated to 100 sessions. With CICS TG V9.0, or later, the ECI connection factory allows the number of sessions to be defined, through the custom property `ipicSendSessions`.

High availability options

Remote mode configurations can use multiple Gateway daemon address spaces, within one or more LPARs, to form a highly available Gateway group. A single port of entry can be used to balance connections between client applications and Gateway daemons, allowing the group of Gateway daemons to act as a single logical Gateway daemon. Such a configuration can provide redundant capacity to take up extra workload in the event of a planned or unplanned outage. It can also be used as a pattern to scale capacity. IPIC connectivity can be used in such configurations.

The TCPIPService listener port for an IPIC connection between CICS TG and CICS TS for z/OS must not be used in conjunction with any TCP/IP port sharing technology for the following reasons. First, XA transaction recovery must be completed with the same CICS server instance, and so reconnection with the same CICS region must be assured. Also, CICS TS for z/OS V4.1 delivered scalability improvements, using multiple TCP/IP sockets for each IPIC connection, and these related sockets must connect to the same CICS server instance.

IPIC connections can be used with any of the available Dynamic Server Selection capabilities available in the CICS TG products. However, individuals, who want to migrate from EXCI and who used the DFHXCURM exit program, must convert to either a DSSPOLICY definition, or implement equivalent logic in the Java-based CICS Request Exit.

Timeout behavior

IPIC connectivity caters for request timeout at both the *application request* level and at a *connection establishment* level.

ECI application programs can optionally specify a timeout value on a request-by-request basis. If this value expires before a response is received from the CICS server, the application receives a timeout error response. The mirror transaction in the CICS server is abended, or marked to abend when it completes, depending on whether it is defined with the `SPURGE` attribute set to `YES`.

CICS TG V9.0 introduced an alternative ECI request timeout option for `IPICSERVER` definitions, `ecitimeout`. This is designed as a safety mechanism, allowing system administrators to impose an overriding time limit for ECI requests, particularly where application code does not specify a sensible timeout value. This is especially useful for those migrating from EXCI, where the granularity for timeout was at the level of a Gateway daemon, or WebSphere Application Server instance (through the DFHXCOPT table). It supersedes any application-coded ECI timeout value at run time.

The server retry interval specifies the time in seconds between attempts to reconnect to a CICS server over an IPIC connection, following a loss of connectivity. This capability is available only in remote mode, and it is predefined to be inactive for local mode IPIC connections.

The server idle timeout value specifies the period of inactivity, after which an IPIC connection and the CICS server are closed. This capability can be configured in remote mode, but it is disabled for local mode IPIC connections.

Note: The default value for server idle timeout on IPIC connections, local and remote mode, changed from 60 to 0 seconds (zero means disabled) starting from CICS TG V9.0.

The connect timeout value specifies the time to wait for CICS to respond during connection establishment. A higher value might be inappropriate for IPIC connections that participate in a high availability group, because it is preferable to fail over quickly when a CICS server becomes unavailable.

Autoinstall

Specific IPIC connection resources (IPConns in CICS TS and Communication Definitions in TXSeries) can be dynamically created at run time, rather than explicitly defined and installed, one by one. This is often useful when connections with many partner systems are required, or short-lived connections, such as might be required in a development or test environment.

The following description of autoinstall applies to the CICS TS for z/OS product.

When an IPIC `TCPIPService` is defined, the Urm attribute is set to `DFHISAIP`, by default. `DFHISAIP` is a CICS-supplied, pre-built autoinstall user program. When a partner system, such as CICS Transaction Gateway, attempts to establish an IPIC connection with CICS TS for z/OS, it looks for a matching `IPConn` definition to use. If no matching `IPConn` definition is found, the autoinstall user program is invoked to generate an `IPConn` definition automatically.

The autoinstall of IPConn resources can be disabled in CICS TS for z/OS, by specifying the value `No` in the Urm attribute of the `TCPIPService` definition.

For further details about using IPIC autoinstall with CICS TG and WebSphere Application Server, see 4.4, “IPIC autoinstall for WebSphere Application Server clusters” on page 64.

For a good overview of IPIC autoinstall with CICS TS for z/OS, see the Information Center topic, *Autoinstalling IPIC connections; preliminary considerations*:

http://ibm.co/1rxNyVn

4.3.2 EXCI

This topic describes characteristics of CICS connections using the EXCI protocol, from the perspective of the CICS Transaction Gateway.

Key differentiators

EXCI connectivity is the original protocol used by CICS TG for z/OS to connect with CICS TS for z/OS. For this reason, it offers limited qualities of service due to architectural constraints:

- Available only on the z/OS platform, in remote and local mode configuration.
- Support for ECI applications is provided:
  - No channel and container support. Only the COMMAREA convention is supported for exchanging data with CICS programs, and so application requests are constrained to the 32-KB architectural limit.
  - Limited timeout capability.
- No support for EPI and ESI applications.
- Password phrases are supported for ECI requests.
- XA transaction support.

- Performance with EXCI can be favorable when compared like for like with IPIC, in terms of overall transaction CPU cost.

  The potential for offload to zAAP engines is limited compared to IPIC, because a significant portion of the implementation is ineligible.
- Scalability of EXCI is constrained by the architectural limitations of EXCI pipes.

Transactions

EXCI connections can be used for synconreturn, extended LUW, and XA transaction-based ECI requests. The CICS TG for z/OS product supports XA transactions when the ECI resource adapter is deployed in local mode, for example, with WebSphere Application Server for z/OS, or remote mode, for example, with a certified Java EE application server.

Note: If you are using extended LUW and XA transaction ECI requests on EXCI connections in a sysplex environment, the CICS Transaction Gateway instance must run on the same LPAR as the CICS server. This is a CICS TS for z/OS limitation.

Security

The security options available for CICS TG configurations using EXCI connections are less complex than IPIC, but effective. Because EXCI uses cross-memory communications, there is no need to consider encryption techniques, such as SSL, and the Gateway daemon never supplies a password or password phrase credential to CICS.

Bind time security, also called *multiregion operation* (*MRO) bind security*, verifies that the system wanting to connect to CICS is authorized to do so. It is implemented using two `DFHAPPL FACILITY` class profiles: `DFHAPPL.<DFHJVPIPE>` and `DFHAPPL.<APPLID>`.

Link security is an additional level of authorization checking applied to the attaching transaction. A specific user ID (the *link user ID*) can be thought of as the user ID associated with the partner system sending a request to CICS. This can be explicitly defined within the CICS region SESSIONS definition for the EXCI connection; otherwise, the user ID of the Gateway daemon address space is used. Both the link user ID and the flowed user ID must be authorized to access all transactions and resources invoked as a result of the request.

User security causes a check to be made against the flowed user ID when an inbound request attaches the transaction in CICS. For ECI requests, the received security context is usually specified by the ECI application, or for Java EE applications, by the ECI connection factory custom property, `userName`. With EXCI connections, only the user ID credential is flowed to CICS TS for z/OS. It is assumed that the user ID has already been authenticated, either by the Gateway daemon in remote mode, or in WebSphere Application Server for z/OS, in local mode.

Surrogate user checking is another security mechanism available for EXCI connectivity, which is useful when the EXCI client (Gateway daemon or WebSphere Application Server) address space user ID is the same as the target CICS region user ID. This might be true when using a common user ID for started procedures.

In such a case, the link security check is bypassed and so a surrogate user check is advised. A surrogate user check is performed to verify that an EXCI client address space user ID is authorized to issue DPL calls for another user (that is, is authorized as a surrogate of the user ID specified on the ECI request).

For full details about EXCI security with CICS TS for z/OS, see the Information Center topic, *Implementing EXCI security*:

http://ibm.co/1n2j4cW

Capacity limits

Capacity for EXCI connections is constrained by an architectural limit of CICS TS for z/OS, imposed on each EXCI client address space. Each ECI request uses a single EXCI pipe, and the maximum number of pipes available per address space is governed at the LPAR level, by the **LOGONLIM** parameter. **LOGONLIM** has a default value of 100, and a maximum configurable value of 250.

Running a Gateway daemon with EXCI enabled effectively constrains its overall capacity, by imposing the same restriction (**LOGONLIM**) on the maximum number of worker threads.

CICS TG for z/OS offers two usage models for EXCI pipes. The pipe reuse model known as ALL allows worker threads to cache pipes, offering improved performance, but compromising concurrent capacity when connecting with multiple CICS TS for z/OS regions. To avoid the possibility of running out of EXCI pipes, the number of concurrent requests must be constrained to **LOGONLIM** divided by the possible number of CICS regions available for connection.

The EXCI pipe reuse model known as ONE does not cache pipes to worker threads and so allows full exploitation of the maximum number of EXCI pipes defined by **LOGONLIM**.

Each EXCI connection is configured to have a specific number of logical sessions available, and it is limited by the **RECEIVECOUNT** parameter defined in the CICS MRO SESSIONS definition. This needs to be at least equal to the maximum number of worker threads in a connected Gateway daemon or the pool size for local mode connection factories in WebSphere Application Server for z/OS.

High availability options

Remote mode configurations can use multiple Gateway daemon address spaces, within one or more LPARs, to form a highly available Gateway group. A single port of entry can be used to balance connections between client applications and Gateway daemons, allowing the group of Gateway daemons to act as a single logical Gateway daemon. Such a configuration can provide redundant capacity to take up extra workload in the event of a planned or unplanned outage. It can also be used as a pattern to scale capacity. EXCI connectivity can be used in such configurations.

In remote mode, EXCI connectivity can also be used with any of the Dynamic Server Selection capabilities available in the CICS TG for z/OS product. However, it is a preferred practice to avoid the use of a Dynamic Server Selection function of the Gateway daemon, together with the EXCI exit program, DFHXCURM. This combination will lead to confusion because the target CICS system selected by the Gateway daemon can be subsequently modified by DFHXCURM. Gateway daemon monitoring and statistics values will be ambiguous because the usage of DFHXCURM is transparent to the Gateway daemon.

Local mode configurations with WebSphere Application Server for z/OS can use a cluster of servant regions for high availability and use EXCI connectivity. However, each servant region is also subject to the **LOGONLIM** capacity constraint, and must use a dedicated, specific EXCI connection to avoid sharing the limited number of sessions (RECEIVECOUNT) in the target CICS region.

> **Note:** CICS Transaction Gateway capabilities for Dynamic Server Selection are not available in local mode, but the EXCI user exit DFHXCURM can be used.

For a more thorough description of workload management within the CICS TG, see 7.3, “Dynamic server selection” on page 116.

Timeout behavior

The granularity of the possible timeouts in an EXCI connection is limited. You can set a TIMEOUT in the EXCI options table DFHXCOPT. This setting is then used by the EXCI to terminate any requests that have been waiting too long for a **DPL_Request** command to complete. It is also possible to have a transaction timeout (such as RTIMEOUT and DTIMEOUT) but it is not an end-to-end timeout.

Autoinstall

There is no autoinstall of EXCI MRO connections.

Although there is no autoinstall capability for EXCI, generic connections offer a useful alternative in development and test environments. Using a generic EXCI connection (`CONNTYPE: GENERIC`) allows Gateway daemon, WebSphere Application Server, and EXCI batch jobs to share a single connection, avoiding the need to install individual connections. However, be careful to ensure that an adequate number of sessions are available, through the associated SESSION definition attribute, `RECEIVECOUNT`.

Because it is not possible to reserve a subset of sessions within the generic EXCI connection for a particular EXCI client, they are not advised for use with CICS TG for z/OS configurations in production.

4.3.3 TCPIP

This topic describes characteristics of CICS connections using the TCPIP protocol, from the perspective of the CICS Transaction Gateway.

Key differentiators

TCPIP connectivity is the original CICS Family protocol to operate on TCP/IP-based networks. For this reason, it has some limitations when compared with IPIC:

- Available only with the CICS TG for Multiplatforms and CICS TG Desktop Edition products, in remote and local mode configuration
- Provides connectivity to all supported CICS servers
- Provides ECI capability with all supported CICS servers

  No channel support. Only the COMMAREA convention is supported for exchanging data with CICS programs, and so application requests are constrained to the 32-KB architectural limit.
- Provides EPI capability with TXSeries and CICS TS for i
- No support for ESI
- No SSL - encryption only possible by using virtual private network (VPN) solutions

Transactions

TCPIP connections can be used for synconreturn and extended LUW transaction ECI requests to all supported CICS server products. There is no support for XA transactions.

Security

TCPIP connectivity to CICS servers offers the following security options:

- *Bind time security* can be used to prevent unauthorized remote systems from connecting to CICS. There is no specific method for implementing bind time security with TCPIP. However, the goal of bind time security can be achieved using standard TCP/IP firewall solutions.
- *Link security* is not supported for TCPIP CICS server connections.
- *User security* can be used to check the received user ID and password that flow from the requestor and determine the user ID under which the transaction operates. This can be used by the CICS server to restrict the CICS resources that can be accessed by a user. For ECI requests, the received user ID and password are usually specified by the ECI application. Or for Java EE applications, it can be specified in the ECI connection factory custom properties: `userName` and `password`. For EPI requests, the user ID and password can be set by the user application at the terminal or connection level, with the value for the terminal taking precedence. Or for Java EE applications, it can be specified in the EPI connection factory custom properties: `userName` and `password`.

With CICS TS for z/OS, there are only two options for user security with the TCPIP protocol:

- No security:
  - No authentication takes place.
  - Transactions are attached in CICS under the authority of the CICS default user.
- User ID and password authentication:
  - The flowed user ID and password are authenticated by the External Security Manager.
  - Transactions are attached in CICS under the authority of the flowed user ID.

There is no capability to attach transactions under the authority of the flowed user ID without a password credential.

Capacity limits

Unlike the other CICS communication protocols available for use with the CICS TG products, the TCPIP protocol is not constrained by a maximum number of logical sessions associated with the connection definition.

TCPIP connectivity is provided by the Client daemon component of both the CICS TG for Multiplatforms and the CICS TG Desktop Edition products. The maximum number of concurrent requests is limited by the maximum number of requests that the Client daemon is configured to process, across all defined CICS server connections. This is determined by the Client daemon configuration option, `MAXREQUESTS`.

High availability options

Remote mode applications using TCPIP connections can use the Dynamic Server Selection capabilities available in the Gateway daemon, the default CICS server definition, and the CICS request exit.

For CICS TG for Multiplatforms or CICS TG Desktop Edition on the Windows platform, an additional option, for both remote and local mode applications, is the Windows Workload manager. This capability allows a set of TCPIP connections to be treated as a single logical CICS server connection, with the option to provide weighting on the distribution of requests.

However, it is a preferred practice to avoid the use of a Dynamic Server Selection function of the Gateway daemon, together with the Windows Workload manager capability. This combination will lead to confusion because the target CICS system selected by the Gateway daemon can be subsequently modified by the Windows Workload manager feature.

Timeout behavior

TCPIP connectivity caters for request timeout at both the application request level and at a connection establishment level.

ECI application programs can optionally specify a timeout value. If this value expires before a response is received from the CICS server, the application receives a timeout error response. The mirror transaction in the CICS server is abended, or marked to abend when it completes, depending on whether it is defined with the SPURGE attribute set to `YES`.

The server retry interval specifies the time in seconds between attempts to reconnect to a CICS server over a TCPIP connection, following a loss of connectivity. Unlike IPIC connections, which allow a server retry interval to be defined on a per-connection basis, TCPIP (and SNA) connections share a single value, defined at the Client daemon level.

The server idle timeout value specified on a server definition for the TCPIP connection protocol specifies the period of inactivity, after which a TCPIP connection to a CICS server is closed.

The connect timeout value specifies the time to wait for CICS to respond during connection establishment. This capability is defined on a per-connection basis. A higher value is might be inappropriate when using Dynamic Server Selection because it is preferable to fail over quickly when a CICS server becomes unavailable.

EPI application programs can optionally specify a terminal install timeout value, and a read timeout value. The install timeout controls the time to wait for a terminal to be installed on a CICS server. If a response is not received from the server within the specified time, control is returned to the user application with an appropriate return code. The read timeout value controls the maximum length of time that the CICS Transaction Gateway will wait for a response from the user application after it has issued an **`EXEC CICS RECEIVE`** or **`CONVERSE`** command.

Autoinstall

For connections to TXSeries using the CICS Family TCPIP protocol, connections can be autoinstalled. For connections to CICS TS for z/OS or CICS TS for z/VSE using the CICS Family TCPIP protocol, connections must be predefined.

4.3.4 SNA

This topic describes characteristics of CICS connections using the SNA (APPC) protocol, from the perspective of the CICS Transaction Gateway.

Key differentiators

CICS connections using the SNA protocol have these differentiators:

- Available only with the CICS TG for Multiplatforms and CICS TG Desktop Edition products, in remote and local mode configuration.
- Support for ECI, ESI, and EPI.

  No channel support with ECI. Only the COMMAREA convention is supported for exchanging data with CICS programs, and so application requests are constrained to the 32-KB architectural limit.
- It requires an additional software product to provide SNA connectivity, such as IBM Communications Server for Data Center Deployment.

  The CICS TG for Multiplatforms and CICS TG Desktop Edition products, up to and including V9.0, require 32-bit SNA API libraries. Some SNA server products might install 64-bit SNA API libraries on 64-bit platforms, by default. For more information about how to install 32-bit SNA API libraries on 64-bit platforms, see your SNA server documentation.
- SNA APPC is not supported between CICS TG products and TXSeries. Use IPIC or TCPIP.
- It is possible to set up a connection using Enterprise Extender, which provides an SNA transport over TCP/IP, where the LU6.2 protocol is encapsulated in TCP/IP packets for transmission across the consolidated IP backbone.

  Example configurations using Enterprise Extender are in the IBM publication, *Migrating an SNA connection from TCP62 to Enterprise Extender,* GC34-6889-00.

For supported products and levels, see the *Supported Software* topic in the Information Center of CICS TG for Multiplatforms.

Transactions

SNA connections can be used for synconreturn and extended LUW transaction ECI requests to all supported CICS server products. There is no support for XA transactions.

Security

SNA APPC connectivity to CICS servers offers the following security options:

- *Bind time security* can be used to prevent an unauthorized system from binding a session to CICS. A security check can be applied when a request to establish an APPC session is received from, or sent to, a remote system; that is, when the session is bound. In SNA terms, this is known as *session security.*

  With CICS TS for z/OS, SNA APPC bind time security is implemented using a combination of CICS system initialization parameters (**SEC**=YES and **XAPPC**=YES), with the SAF profile, APPCLU.
- *Link security* is an additional level of authorization checking applied to the attaching transaction. A specific user ID (the *link user ID*) can be thought of as the user ID associated with the partner system sending a request to CICS.

  With CICS TS for z/OS, this can be explicitly defined within the CICS region CONNECTION definition (SECURITYNAME), or SESSION definition (USERID) for the SNA APPC connection. If you do not specify a link user ID, CICS TS for z/OS uses the CICS default user ID. Both the link user ID and the flowed user ID must be authorized to access all transactions and resources invoked as a result of the request.

- *User security* can be used to check the received user ID and password that flow from the requestor and determine the user ID under which the transaction operates. This can be used by the CICS server to restrict the CICS resources that can be accessed by a user. For ECI requests, the received user ID and password are usually specified by the ECI application. Or for Java EE applications, it can be specified in the ECI connection factory custom properties: `userName` and `password`. For EPI requests, the user ID and password can be set by the user application at the terminal or connection level, with the value for the terminal taking precedence. Or for Java EE applications, it can be specified in the EPI connection factory custom properties: `userName` and `password`.

With CICS TS for z/OS, there are only two options for user security with the SNA APPC protocol:

- No security:
  - No authentication takes place.
  - Transactions are attached in CICS under the authority of the CICS default user.
- User ID and password authentication:
  - The flowed user ID and password are authenticated by the External Security Manager.
  - Transactions are attached in CICS under the authority of the flowed user ID.

There is no capability to attach transactions under the authority of the flowed user ID without a password credential.

SNA APPC supports ESI requests, allowing application programs to perform password expiry management (PEM) operations, including change and verify password.

For full details about SNA APPC security with CICS TS for z/OS, see the Information Center topic, *Implementing LU6.2 security*:

http://ibm.co/1szvVqU

Capacity limits

The workload capacity for SNA APPC connectivity is primarily governed by sessions.

With CICS TS for z/OS, the SESSIONS definition (`MAximum`) is configured to specify the maximum number of sessions to be supported, and the number of contention-winner sessions (CICS-owned sessions). These values are negotiated during change number of sessions (CNOS) flows, when the sessions are actually bound; the negotiated values depend on the settings specified in the partner SNA node.

The maximum number of sessions must be at least as big as the **`MAXREQUESTS`** parameter in the `CLIENT` section of the CICS TG configuration file, to prevent creating a bottleneck to throughput. The contention-winner sessions must be set to 001 to ensure that `START` requests for terminal transactions are shipped serially from the server to the client.

With CICS TG for Multiplatforms, or CICS TG Desktop Edition, the partner SNA node is configured through an SNA server product, such as IBM Communications Server, where the number of sessions is defined in relation to the mode name for the APPC connection.

The maximum number of concurrent requests is also limited by the maximum number of requests that the Client daemon is configured to process, across all defined CICS server connections. This is determined by the Client daemon configuration option, `MAXREQUESTS`.

High availability options

SNA APPC connectivity can exploit the z/OS Communications Server generic resource function to balance workload between available CICS TS for z/OS regions. CICS TS for z/OS regions are assigned two APPLIDs, a generic one and a specific one. The IBM VTAM® component of z/OS Communication Server maintains a mapping between the generic APPLID, and the specific CICS APPLIDs that are members of a generic resource name.

The CICS TG for Multiplatforms or CICS TG Desktop Edition product is configured with an SNA connection, using the generic CICS APPLID. This also allows VTAM to perform dynamic workload balancing of the sessions across the available CICS TS for z/OS regions.

Remote mode applications using SNA APPC connections can exploit the Dynamic Server Selection capabilities available in the Gateway daemon, the default CICS server definition, and the CICS request exit.

For CICS TG for Multiplatforms or CICS TG Desktop Edition on the Windows platform, an additional option, for both remote and local mode applications, is the Windows Workload manager. This capability allows a set of SNA APPC connections to be treated as a single logical CICS server connection, with the option to provide weighting on the distribution of requests.

However, it is a preferred practice to avoid using the Dynamic Server Selection function of the Gateway daemon together with the Windows Workload manager capability. This combination will lead to confusion because the target CICS system selected by the Gateway daemon can be subsequently modified by the Windows Workload manager feature.

Timeout behavior

SNA APPC connectivity caters for request timeout at the application level and at the configuration level.

ECI application programs can optionally specify a timeout value. If this value expires before a response is received from the CICS server, the application receives a timeout error response. The mirror transaction in the CICS server continues to runs until the CICS program ends.

For synconreturn requests, the CICS program will complete its syncpoint normally, irrespective of the ECI timeout value, and the mirror transaction ends. For extended LUW requests, the mirror transaction is suspended, waiting for a further request. If the mirror transaction PROFILE is defined with a suitable read timeout value (`RTIMOUT`) and is defined with the SPURGE attribute set to `YES`, it will abend due to read timeout expiration.

The server retry interval specifies the time in seconds between attempts to reconnect to a CICS server over an SNA connection, following a loss of connectivity. Unlike IPIC connections, which allow a server retry interval to be defined on a per-connection basis, SNA (and TCPIP) connections share a single value, defined at the Client daemon level.

The server idle timeout value specified on a server definition for the SNA connection protocol specifies the period of inactivity, after which an SNA connection to a CICS server is closed.

EPI application programs can optionally specify a terminal install timeout value, and a read timeout value. The install timeout controls the time to wait for a terminal to be installed on a CICS server. If a response is not received from the server within the specified time, control is returned to the user application with an appropriate return code. The read timeout value controls the maximum length of time that the CICS Transaction Gateway will wait for a response from the user application after it issues an **`EXEC CICS RECEIVE`** or **`CONVERSE`** command.

Autoinstall

Specific SNA APPC connection resources can be dynamically created at run time, rather than explicitly defined and installed, one by one. This is often useful when connections with many partner systems are required, or short-lived connections, such as might be required in a development or test environment.

The following description of autoinstall applies to the CICS TS for z/OS product.

The CICS ISC capability must be enabled, if not already (**ISC**=YES). You must select your preferred autoinstall program. The CICS-supplied autoinstall exit program, DFHZATDY, can be used to provide APPC autoinstall (**AIEXIT**=DFHZATDY). The correct 'template' connection that will be used to create the new connection must also be installed.

When an APPC device attempts to log on to the CICS TS for z/OS region, a check for a matching APPC connection definition is made. If no matching definition is found, the autoinstall program is driven to generate a definition, based on the closest matching template definition.

Note: CICS TS for z/OS can be configured with a maximum number of concurrent logon requests (AIQMAX), which can result in the temporary suspension autoinstall activity.

4.4 IPIC autoinstall for WebSphere Application Server clusters

When using the CICS ECI resource adapter in a local mode topology from WebSphere Application Server to establish an IPIC connection to CICS TS, it is a preferred practice to specify APPLIDQUALIFIER and APPLID values in the Java EE Connector architecture connection factory custom properties, to identify the source of requests in monitoring data.

With a single application server configuration, the IPConn resource in CICS TS can be predefined or autoinstalled. However, if you are using a WebSphere Application Server cluster configuration, autoinstalled connections, which use the CICS default IPIC autoinstall user program, DFHISAIP, or predefined IPIC connections will encounter problems due to duplicate client identifiers.

The problem occurs when a Java EE application uses a cluster scope Java EE Connector architecture connection factory or multiple Java EE Connector architecture connection factory definitions with the same APPLIDQUALIFIER and APPLID values and displays the following symptoms:

1. The first application server in the cluster sends an ECI request to CICS TS causing an IPIC connection to be installed (either predefined or autoinstalled) and acquired.
2. Then, a different application server sends another ECI request. CICS TS tries to use the same IPCONN resource definition already installed, because the APPLIDQUALIFIER and APPLID values match, but rejects the request because that IPCONN resource is still in the acquired state by the first application server. The following error message DFHIS1015 is issued to the MSGUSR job log of the CICS TS region:

   “Unable to accept connection for IPCONN *ipConnName*. IPCONN client session state is invalid.”

One solution to this problem is to leave the APPLIDQUALIFIER and APPLID values in the connection factory blank. In that case, DFHISAPI will generate values upon connection. However, this makes transaction tracking difficult because the generated values are transitory and will not be so useful for audit or debug purposes.

CICS TS 5.1 provides a better solution to this limitation through new sample IPIC autoinstall programs `DFH$ISAI` (Assembler) and `DFHOISAI` (COBOL). These create IPCONN resources based on a template `IPCONN` resource definition, whose name and Applid attribute values must match the APPLIDQUALIFIER and APPLID of the partner respectively. If the connection request is accepted, the autoinstalled `IPCONN` name and its Applid attribute value will be dynamically generated using the partner APPLID followed by a unique integer suffix. The Networkid attribute will be set to the APPLIDQUALIFIER.

For example, assume the following definitions:

- Java EE Connector architecture connection factory custom properties:
  - APPLIDQUALIFIER value `CTGAPQAL`
  - APPLID value `CTGAPPL`
- CICS `TCPIPSERVICE` resource definition attributes:

  `URM` value `DFHOISAI`
- CICS IPCONN resource definition template called `CTGAPQAL` with attribute:

  Applid value `CTGAPPL`

The following steps show the behavior for connecting from a WebSphere Application Server cluster:

1. The first IPIC connection request from an application server to that CICS TS region results in an IPCONN resource definition called `CTGAPPL1` with an Applid attribute value of `CTGAPPL1` and Networkid attribute value of `CTGAPQAL` being installed.
2. The next IPIC connection request to that CICS TS region from a different application server instance results in an `IPCONN` resource definition called `CTGAPPL2` with an Applid attribute value of `CTGAPPL2` and Networkid attribute value of `CTGAPQAL` being installed.
3. Subsequent connection requests from the same application server will reuse the `IPCONN` resource installed by the first request from that application server and will remain installed and acquired until the application server is shut down.

Note: The alternative Urm appends a counter value to the APPLID value in the connection factory. Because the maximum length of the APPLID attribute is eight characters, the value set in the connection factory determines the maximum possible number of application server instances able to connect with a common identification.

The sample autoinstall user programs `DFH$ISAI` and `DFHOISAI` are provided as precompiled load modules in the load library, *`CICSHLQ`*`.SDFHLOAD`, and can be used directly, provided that the APPLID value in the ECI connection factory definition is no more than seven characters in length. Alternatively, the samples can be customized using the source code provided in the partitioned data set, *`CICSHLQ`*`.SDFHSAMP`. For more information, see the CICS TS Information Center topic, *Sample autoinstall user program to support predefined connection templates*:

http://ibm.co/1wG5Rbh

The CICS TG Information Centers include several scenarios that demonstrate how to configure IPIC connections:

- SC01. Configuring a secure autoinstalled IPIC connection
- SC02. Configuring a secure predefined IPIC connection
- SC07. Configuring SSL between CICS TG and CICS TS
- SC08. Configuring an autoinstalled IPIC connection

5

Understanding the Gateway daemon

CICS TG is implemented in remote mode when the client application and CICS TG are running on different machines and the Gateway daemon listens on a specific port for incoming client requests. This is also called a 3-tier solution.

In a remote mode configuration, the CICS Transaction Gateway runs a process known as a *Gateway daemon,* which listens on a specific port, receives requests from remote client applications, forwards those requests to specific CICS servers, and communicates the result back to the remote client application.

This chapter provides insight into the Gateway daemon process and allows you to understand the implication of different Gateway daemon configurations. It also describes how to influence the management of connections and thread resources in the Gateway daemon.

It also provides insight into the system dependencies of the Gateway daemon configuration, allowing you to perform basic tuning suggestions to improve Gateway daemon performance.

All the formulas that are provided are based on observations, and they provide a general guideline to configure the system. Every configuration has different requirements, and so your configuration must be verified with the Gateway statistics or the operating system tools. Some values that are provided are subject to change, based on the level of the operating system where the Gateway daemon is running.

The following topics are covered in this chapter:

5.1 Three-tier concepts

CICS TG can be used in two modes: local and remote. There are also a number of possible combinations of implementing the networking and system infrastructure between the client applications, CICS TG components, and CICS servers. The different combinations are referred to as *topologies*.

Remote mode operation uses the Gateway daemon as a long-running process. The Gateway daemon listens on a specified port for incoming requests (using a TCP/IP connection), then it forwards those requests to a CICS server using one of the listed protocols: EXCI, IPIC, TCPIP, or SNA.

Figure 5-1 shows different topologies. Topology 2 and 3 show a remote mode configuration where the Gateway daemon is representing the middle tier of a 3-tier solution.

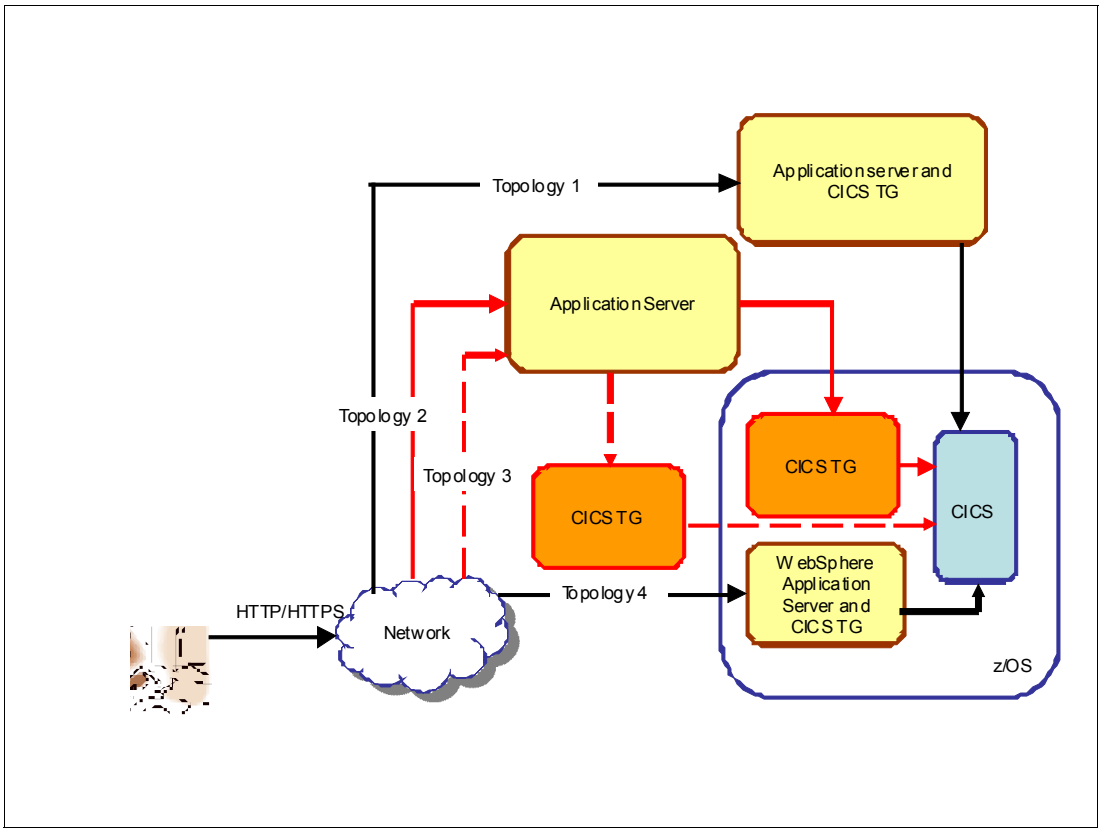


Figure 5-1 CICS TG topology examples

The client applications, using a CICS TG API, are able to invoke CICS programs through the Gateway daemon. However, the connection between the client applications and the Gateway is not always implicitly created. ECI V2, ESI V2, .NET, and Java base class applications need to manage the connection by using explicit open and close, API calls. Java EE application servers have managed connection factories, with a pool of cached connections already created.

Figure 5-2 on page 69 shows the connections management based on the APIs and technology used.

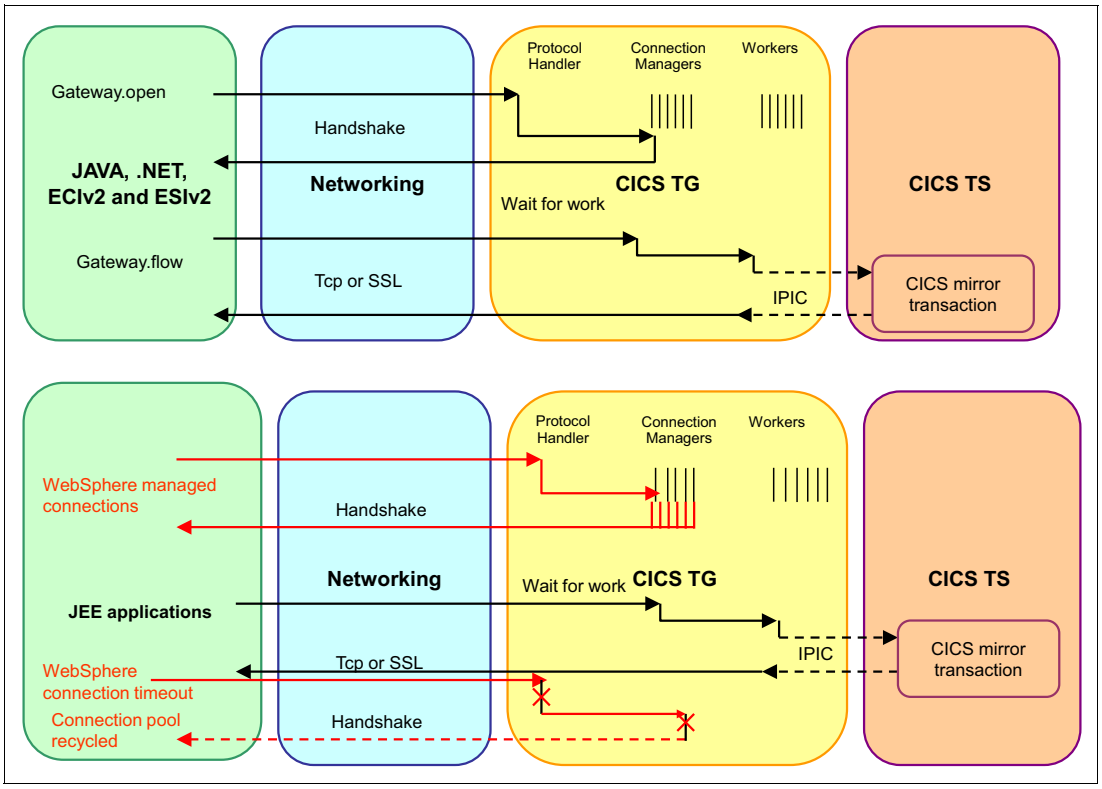


Figure 5-2 Flow of control through a Gateway daemon

The Gateway daemon is designed to act as a network concentrator, able to serve potentially many remote applications through relatively restricted CICS connections. The client can connect in various ways. When the Gateway daemon uses IPIC connections, almost all of the run time takes place with the Java virtual machine (JVM). Other protocols require additional native runtime components as shown in Example 5-2 on page 85.

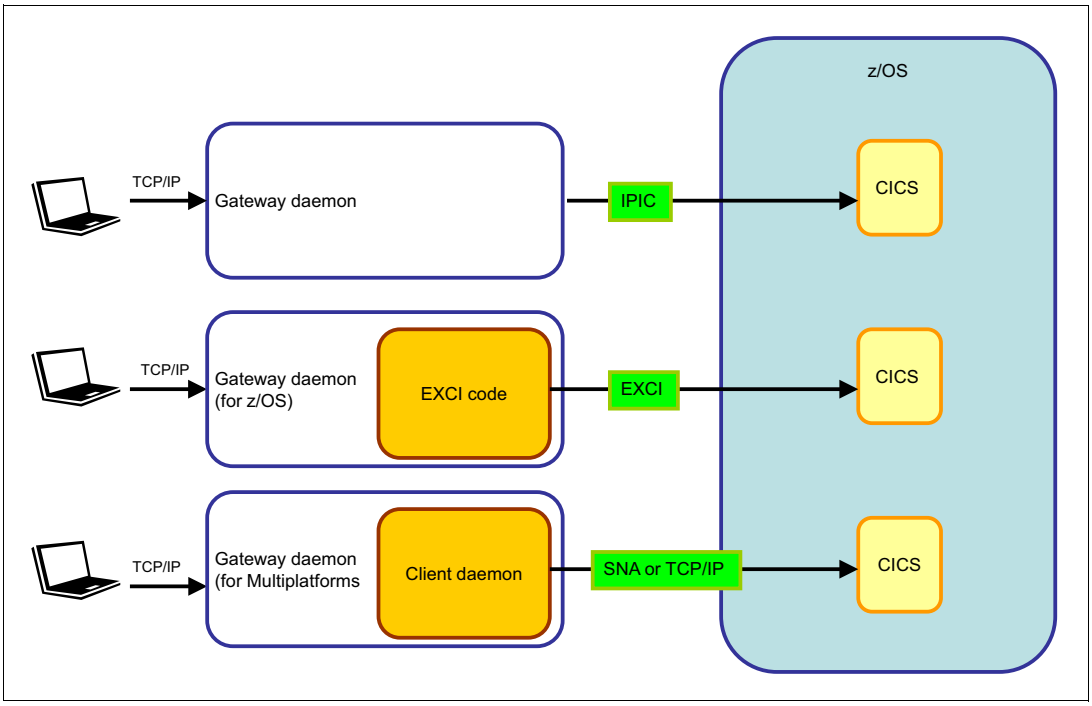


Figure 5-3 CICS TG components related to CICS communication protocols

5.2 Gateway daemon resources

This section describes the resources that the Gateway daemon needs to satisfy the client requests.

5.2.1 Protocol handlers

This section discusses the protocol handlers.

TCP and SSL

The Transmission Control Protocol (TCP) and Secure Sockets Layer (SSL) protocol handlers represent the entry point to the Gateway daemon for remote client applications. The Gateway daemon provides two connection protocols: TCP and SSL. Use SSL connectivity when the data between the client and the Gateway daemon must be secure. Table 5-1 summarizes the API restrictions using SSL.

Table 5-1 Remote API components and connection protocol support

Remote mode applications	TCP	SSL
All APIs in Java	Yes	Yes
All APIs in JCA	Yes	Yes
.NET	Yes	With SupportPac CA76[a]
ECI v2	Yes	No
ESI v2	Yes	No

a. Download available at http://www.ibm.com/support/docview.wss?uid=swg24034001

Protocol handlers are defined in the `GATEWAY` section of the configuration file. You need to define the port where the Gateway will listen, and the **`bind`** parameter if you have more than one active IP address.

> **Note:** You can have at most one TCP and one SSL handler.

To verify that SSL security is enabled on the Gateway daemon, check the following points:

- Whether the SSL ciphers have been enabled in the Gateway daemon. The following message is written to the Gateway daemon information log:

  ```
  CTG8489I The following cipher suites are provided by JSSE: with all the cipher
  suites supported
  ```

- If the SSL protocol handler is successfully started on the configured port, you will see the following message in the Gateway daemon information log:

  ```
  CTG6524I Successfully started handler for the ssl: protocol on port 8573
  ```

For additional clarification about security, see Chapter 8, “Security” on page 131, and for implementation, see 10.4, “Configure secure connections” on page 186.

Statistics and administration handlers

On all platforms, CICS TG provides statistics information about the status and the activities performed by the Gateway daemon. It can help you in tuning the Gateway daemon configuration and for post-processing analysis of the information collected by the statistics engine of the Gateway daemon. To enable the statistics, define the *statsapi* statistics API protocol handler, in the `GATEWAY` section of the CICS TG configuration file.

You can display the statistics using the administration interface (CICS Explorer for example) or retrieve the information using statistics APIs in a user program.

To verify that the *statsapi* port is enabled on the Gateway daemon, check the following message in the Gateway daemon information log:

```
CTG6524I Successfully started handler for the statsapi: protocol on port 2989
```

If the Gateway daemon is installed on multiplatforms, we need also to define an additional handler for administering the Gateway daemon. When an administrative request is received by a Gateway daemon from the **`ctgadmin`** command, it is processed by a restricted handler. On Multiplatforms, this handler is defined in the Gateway daemon section using the parameter **`ctgadmin`**.

To verify that the admin port is enabled on the Gateway daemon, check the following message in the Gateway daemon error log:

```
CTG8455I Successfully started the local administration handler on port 2810
```

5.2.2 The threading model

The Gateway daemon uses two thread pools to maintain the *link* between the client request and the target CICS. When an application opens a connection to a Gateway daemon, a connection manager thread is allocated. This thread represents the active application connection. When the application sends a flow, the Gateway daemon needs to send the request to CICS, allocating a worker thread to represent the active request to CICS. The Gateway daemon can handle multiple requests simultaneously, at the capacity that is configured in the Gateway daemon configuration file.

Connection manager threads manage the connections from a particular remote client. When a connection manager thread receives a request, it allocates a worker thread from a pool of available worker threads to run the request, and it remains allocated. The worker thread is allocated to process CICS requests on behalf of connection manager threads. When a worker thread finishes the processing, it returns to the pool of available worker threads. The connection manager threads can be deallocated if an explicit close is issued, or it can be just reused by a new request.

For each connected client, one connection manager thread is used in the Gateway daemon. This thread is held until the client closes the connection, or the Gateway times out the connection. To perform an ECI request using an allocated connection manager thread, a thread must be allocated from the worker thread pool for the duration of the ECI request.

Figure 5-4 shows the interactions between the different threads.

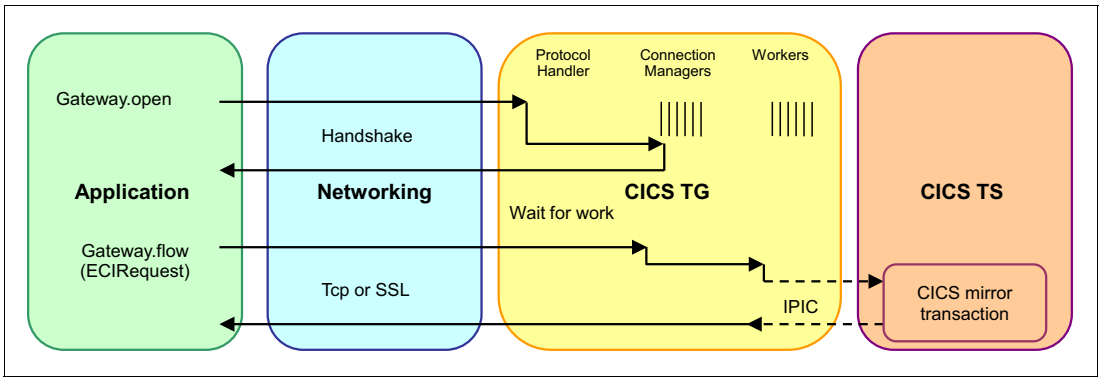


Figure 5-4 The threading model through the Gateway daemon

It is important to consider the thread limits in the system when setting the number of connection manager and worker threads. The limit for the system threads is set so high that it cannot be realistically reached; the real limit is driven by the amount of virtual memory available to the process and by the CPU capacity.

On z/OS, this might be restricted by the total number of MVS Task Control Blocks (one is created for each UNIX System Services thread). This limit is governed by the UNIX System Services parameters **MAXTHREADS** and **MAXTHREADTASKS** in the BPXPRMxx member of `SYS1.PARMLIB`. The Gateway daemon creates the connection manager and worker threads using the UNIX System Services thread facility. The total number of threads in use by the Gateway daemon can be displayed using the MVS system command **/D OMVS,L,PID=*nnnn***, where ***nnnn*** is the process ID of the JVM running the Gateway daemon, as displayed using the SDSF PS menu option. Consider also the UNIX System Services parameter **MAXPROCUSER**. It specifies the maximum number of processes that a single user (same UID) can have concurrently active. If a Gateway daemon is started with a user with a UID of 0, it is not limited by the **MAXPROCUSER** value (a superuser is always allowed to run so that it can be invoked to solve a problem).

To display the current UNIX configuration setting, you can also run the MVS system command **/D OMVS,OPTIONS**.

To temporarily modify the settings for **MAXTHREADS** and **MAXTHREADTASKS**, use the **/SETOMVS MAXTHREADS=*nn*** or **/SETOMVS MAXTHREADTASKS=*nn*** commands. To permanently change the values, modify the BPXPRMxx member.

5.2.3 Estimating the number of connection manager threads

The best way to calculate the number of required connection manager threads depends on whether you are using a Java EE application or a base Java application. With Java EE applications, the number of connection manager threads depends on the size of the connection pool defined in the connection factory of the JCA connection pools.

WebSphere Application Server maintains a pool of open TCP/IP connections to the Gateway daemon. The number of connections in the pool, at any time, depends on the number of concurrent requests for the CICS server defined by the connection factory and how quickly it is configured to close inactive connections.

To avoid a resource shortage when WebSphere Application Server is running under high load, control your setting of **maxconnect**. The **maxconnect** setting is in the Gateway daemon initialization file. It must be greater than or equal to maximum connections in the connection pool. If you are using multiple WebSphere Application Server instances or multiple resource adapters, **maxconnect** must be greater than or equal to the sum of the connections for all of the connection factories.

For remote client applications (Java base classes, ECIv2, and .NET), which must explicitly manage the connections to the Gateway daemon (open-flow-close method), the number of connections in use depends on the design of the user application. For example, the general sequence of events for an application with Java base class is to open a connection using **JavaGateway.open()**, issue any number of requests through **JavaGateway.flow()**, and close the connection using **JavaGateway.close()**. Each instance of a JavaGateway object maintains one TCP/IP socket between an open and a close. The number of connection manager threads must be greater than or equal to the concurrent number of JavaGateway objects that have been opened by the user's applications.

It is possible to configure the Gateway daemon so that an application will wait for a connection manager's threads to become free. The **connecttimeout** parameter in the Gateway daemon configuration file controls this value. Only consider reducing the connection manager threads available in the pool and relying on this timeout if applications open and close connections frequently.

A thorough evaluation of the maximum number and the initial number of connection manager threads avoids redundant storage allocation and any additional CPU workload due to the management of inactive threads. A thorough evaluation of the maximum number and the initial number of connection manager threads can make the CICS TG startup time faster.

In the configuration file, you control the maximum number of connection manager threads with the parameter **maxconnect**.

The configuration parameter, **initconnect**, identifies the number of connection manager threads created at the Gateway daemon startup.

Note: For CICS TG Desktop Edition, 5 is the maximum value of **maxconnect** and **initconnect**.

5.2.4 Estimating the number of worker threads

The number of worker threads depends on the concurrent number of requests from remote clients and on the numbers of CICS sessions available to process the incoming requests. When a worker thread has finished processing the request, it returns to the pool of available worker threads. For this reason, there is no need to code the maximum number of worker threads higher than the maximum of connection manager threads.

When considering the CICS TG maximum number of worker threads, take into account how the following CICS Transaction Server (CICS TS) configuration parameters affect the Gateway daemon:

- **MAXTASKS** in CICS TS: The Gateway daemon cannot start more mirror transactions than CICS TS can concurrently run. If you have more than one CICS TS, you need to consider how the workload is distributed.
- **TRANCLASS** in CICS TS**:** The mirror transactions that the Gateway daemon attaches in CICS TS can be controlled by the number of concurrent running tasks defined in a particular transaction class. You must also take into account the PURGETHRESHOLD value, in the **TRANCLASS** definition.
- Regardless of the solution you decide to use to connect to CICS TS (IPIC, EXCI, or ECI over IP), you cannot exceed the upper limit of the protocol connection definition. This aspect will be described later in this chapter.
- Remember that CICS TS can be configured to have multiple connections with other CICS TS. To process the distributed program link (DPL) requests sent by the Gateway daemon, you might need to send the request to another CICS system, such as an Application Owner Region (AOR). This kind of topology can introduce additional elements that must be considered.
- The application can also invoke some processes in CICS TS that create affinities or need sequential access to resources. Any possible constraint has to be taken into account.

If the number of communication manager threads is higher than the worker threads, it is possible that the Gateway daemon is queuing for free worker threads. You can establish how long your request will queue for an available worker thread by specifying the **workertimeout** parameter in the CICS Transaction Gateway daemon configuration file.

In the configuration file, you control the maximum number of worker threads with the parameter **maxworker**.

The configuration parameter, **initworker**, identifies the number of worker manager threads created at the Gateway daemon during initialization.

> **Note:** For CICS TG Desktop Edition, 5 is the maximum value of the **maxworker** and **initworker**.

5.3 Connecting to CICS servers

This section describes the main implications that must be evaluated when configuring the Gateway daemon depending on the protocols selected to connect to CICS TS.

The storage implications with the different protocols used are expanded in 5.4, “System resources” on page 78.

For selection guidelines based on the protocol that is used to connect the Gateway to a CICS TS, see Chapter 1, “Basic concepts” on page 3 and Chapter 4, “Connecting to CICS” on page 43.

5.3.1 All platforms: IPIC

The Gateway daemon can connect to CICS TS using IPIC protocol, independent of where the CICS TG is installed. It can be secured with SSL.

IPIC provides ECI access to CICS applications over the TCP/IP protocol, supporting both COMMAREA and CICS channel applications.

You can configure the IPIC connection in the `IPICSERVER` section of the configuration file and it needs to match the CICS TS definition. Figure 5-5 shows the relationship between the different elements.

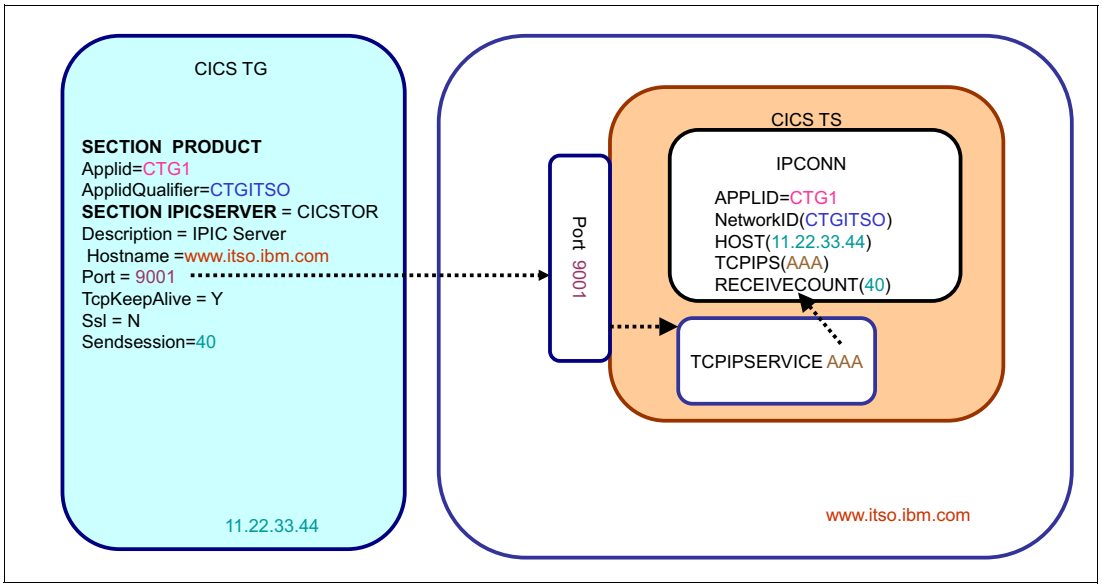


Figure 5-5 Relationship between CICS TG and CICS TS definitions for an IPIC connection

When you are configuring the number of worker threads, you need to take the **`sendsessions`** parameter into account. If you have multiple IPIC server definitions, remember to add the **`sendsessions`** number to the worker thread count to allow the Gateway to manage all the sessions.

The **`sendsessions`** parameter, in the `IPICSERVER` section of the configuration file, specifies the number of simultaneous CICS tasks that are allowed over the CICS connection. You must specify the same number for the value in the **`receivecount`** parameter of the IPCONN definition on the CICS server. If you do not specify a value or specify a lower value in **`receivecount`** than the value specified in the **`sendsessions`** parameter, the value used might be reduced.

You can also find a benefit using the CICS IPCONN autoinstall. During the first connection from CICS TG to CICS, CICS creates the IPCONN definition. Unlike autoinstall for other resources, autoinstall for IPCONN resources does not require model definitions, although they are advised.

Client applications usually flow data in ASCII format and the CICS user program is able to manage data in EBCDIC in the COMMAREA. The CICS mirror program (`DFHMIRS`) invokes the services of the data conversion program (`DFHCCNV`) to perform the necessary code page conversion of the COMMAREA.

The CICS channel and container solution allows you to send and receive more than 32 KB of application data in a single ECI request. This ability requires you to consider additional points when you are configuring your Gateway.

The data conversion model, used by channel applications, is controlled by the application developer using simple API commands, and `DFHCNV` is not needed anymore. CICS TS uses z/OS Unicode Services instead. You can check whether the z/OS Unicode Services are correctly implemented using the following z/OS system command:

```
D UNI,ALL
```

Consider the performance and storage allocation implications of passing a large amount of data through channels. For example, when a large amount of data is to be passed to a remote program or transaction, it might affect performance.

If your applications need to use a container size greater than 1 MB, it might be necessary to increase the JVM resources and the memory size for the Gateway daemon and CICS. A channel might use more storage than a COMMAREA designed to pass the same data because container data can be held in more than one place. COMMAREAs are accessed by pointers. The data in containers is copied between programs.

For the complete list of benefits of using channel and container technology, see the CICS TS Information Center topic, *Benefits of channels.*

IPIC on CICS TG for z/OS specific items

IPIC can benefit from the System z Application Assist Processor (zAAP) because the IPIC workload can be offloaded to these processors. Offloading the IPIC workload significantly reduces the CPU cost per transaction. Although EXCI is limited to 32-K payloads using a COMMAREA, IPIC payloads can be as large as you need, using channels and containers.

When the EXCI layer is loaded, the maximum number of worker threads is capped at 250. Specifying **`CTG_EXCI_INIT`**=`YES` and **`maxworker`** to more than 250 will cause the following message to be issued during the Gateway daemon startup and the value of the **`maxworker`** is reduced to be equal to the EXCI logon limit (**`LOGONLIM`**):

```
CTG6406W the maximum worker threads has been reduced to the EXCI logon limit 250
```

IPIC connections to CICS are not limited by **`LOGONLIM`** because they do not use EXCI pipes. But if the EXCI layer is loaded, it is assumed that the EXCI connections will be used, so the **`maxworker`** restriction is applied. To allow IPIC to use more than 250 worker threads, **`CTG_EXCI_INIT`**=`NO` must be set to prevent the EXCI layer being loaded.

Note: Applications can only use ESI with CICS TG for z/OS when the CICS connection is IPIC.

5.3.2 Multiplatform and Desktop Edition specific: TCPIP

CICS TS has no way to limit the number of concurrent requests that are sent to a TCP/IP connection. Applications in error can submit an excessive number of requests to a server. In this event, to prevent runaway conditions and avoid a possible denial of service attack that can affect CICS TS, it is suggested to code an accurate number of worker threads.

The **`maxrequests`** parameter can also be used to prevent this situation.

> **Note:** TCPIP is the name of a driver type and it is not TCP/IP, the transport.

5.3.3 Multiplatform and Desktop Edition specific: SNA

Using an SNA connection, the **`maxrequests`** parameter in the `ctg.ini` file must be the same or more than the number of sessions defined in CICS.

The first value, in the CICS session parameter, 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 during change number of sessions (CNOS) flows. When the sessions are actually bound, the negotiated values depend on the settings specified in the partner SNA node.

This solution also requires a supported SNA provider product (for example, IBM Communications Server) to be installed and configured in the same machine where CICS TG is installed.

> **Note:** CICS TG Desktop Edition runs only in 2-tier remote mode.

5.3.4 z/OS specific: EXCI

The external CICS interface (EXCI) is used by the Gateway daemon for z/OS to send requests to the CICS TS region using cross-memory services.

To load the code to support the EXCI protocol, you need to set the environment variable **`CTG_EXCI_INIT`** to `YES`. At this point, the Gateway will also load the EXCI options table, `DFHXCOPT`. There is no suffixed version of this program, so the first DFHXCOPT table in the `STEPLIB` concatenation is loaded.

`DFHXCOPT` allows you to specify whether you want the surrogate-user check, whether you want the timeout value for requests over EXCI, and whether the trace is active.

> **Tip:** Specify `ABENDBKOUT=YES` in `DFHXCOPT` if you want to back out updates made by a long-running mirror following a transaction abend.

When a request is sent over an EXCI connection, a pipe is allocated; the Gateway does not perform any deallocate after it flows the initial ECI request for a given worker thread.

An EXCI pipe stays allocated for the lifetime of the thread. An EXCI pipe is only deallocated if one of the following events occurs:

- An error is received by the Gateway worker thread on an **Open_Pipe** call (such as when a CICS connection has been closed).
- The Gateway daemon terminates.
- The worker thread allocates another pipe to a different CICS applid, and the **CTG_PIPE_REUSE** environment variable is set to ONE.

Every address space that uses EXCI pipes, such as a Gateway daemon, is limited to a maximum number of pipes that it can allocate to all attached CICS regions. The EXCI pipe limit is governed by the z/OS system-wide **LOGONLIM** parameter and the upper limit is 250.

Note: To modify the **LOGONLIM**, you need to modify the SYS1.PARMLIB(DFHSSI*xx)* member.

Also, consider that deallocating an EXCI pipe has a cost in terms of CPU consumption. For this reason, it is important to select the best way for your installation to reuse the EXCI pipe.

This is governed by the parameter **CTG_PIPE_REUSE**.

CTG_PIPE_REUSE=ALL means that the pipes will be permanently allocated to a particular CICS server and are not available for use with other CICS servers. It is suggested that **CTG_PIPE_REUSE**=ALL is only specified when the Gateway daemon uses EXCI connections to one CICS server. If you do specify **CTG_PIPE_REUSE**=ALL and EXCI connections to more than one CICS server are defined, ensure that you do not run out of EXCI pipes by limiting the number of worker threads that can allocate the pipes. To ensure that you do not run out of EXCI pipes by limiting the number of worker threads that can allocate the pipes, use this formula to configure the number of worker threads:

```
MaxWorkers <= Pipes Limit / Number of servers
```

CTG_PIPE_REUSE=ONE means that if requests are made to multiple CICS servers, the pipe will be deallocated and another pipe will be allocated, before the call to the second CICS server takes place.

If subsequent requests are made to the same CICS server, no pipe deallocation and reallocation will take place. In this case, consider using the following formula:

```
MaxWorkers <= Pipes Limit
```

For all connected Gateway daemons, on z/OS, the CICS region requires both a CONNECTION definition and a SESSIONS definition. The CONNECTION definition identifies the remote system, and the SESSIONS definition associated with this connection determines the properties of the sessions.

The maximum pipe usage in a CICS region is also limited by the **RECEIVECOUNT** parameter defined in the CICS MRO SESSIONS definition (this can be up to 999).

5.4 System resources

This section describes how the Gateway daemon interacts with system components and important considerations when you configure it.

5.4.1 Gateway daemon runtime environment

During startup, the Gateway daemon uses a combination of the configuration file and an environment variable to form the runtime environment and to successfully start the required processes. When needed, you can also pass some override parameters to the Gateway daemon.

Table 5-2 shows the different resources involved in the Gateway daemon startup.

Table 5-2 Gateway startup resources

Platform	Configuration file location	Environment variable	Override
z/OS	Specified in CICSCLI in the environment variable.	Specified in the DD STDENV	- In the `EXEC PARM` at **`ctgstart`** level or - **`CTGSTART_OPTS`** in the environment variable
Windows	Default to <*product_data_path*>/`ctg.ini`. It can be changed using the **`CICSCLI`** environment variable.	Specify as a system environment variable	- Using the utility **`ctgservice -R`** or - starting the **`cicscli -R`** process
UNIX/Linux	Default to <*install_path*>/`bin/ctg.ini`. It can be changed using **`CICSCLI`** environment variable.	- Specify as environment variable or - in `CTGDCONF`, if **`ctgd start`** is used. The default **`ctgd`** configuration file is <*install_path*>/`bin/ctgd.conf`	- Starting the **`cicscli`** process or - **`CTGD_PARMS`** environment variable

Java Runtime Environment is required to use the CICS TG core components. Use the latest Java update for your Java Runtime Environment (JRE) on Multiplatform and Desktop Edition. For z/OS, use the IBM SDK for z/OS, Java Technology Edition.

The Gateway daemon for z/OS can be started using **`CTGBATCH`**, which invokes the z/OS UNIX script, **`ctgstart`**, as a batch job. It is still possible to start the Gateway in the UNIX System Services environment; however, you cannot benefit from the z/OS facilities, such as the visibility of the logs in System Display and Search Facility (SDSF), z/OS console commands, and Gateway daemon-specific storage size of the address space.

The Gateway daemon for UNIX or Linux can be started as a background process, using the **`ctgd start`** command, and it runs under a specific user ID.

The Gateway daemon for Windows runs as a service that is started manually by default.

5.4.2 Information available at run time

The CICS TG has numerous log files. Table 5-3 on page 80 summarizes where the locations of the log files based on the platform.

The Client daemon log, Gateway daemon info log, and Gateway daemon error logs are available.

Table 5-3 Location of log files

Operating system	Log name	Default location	Override location
Windows	`Client daemon log`	`<product_data_path>` `directory. By default, it is C:\ProgramData\IBM\CICS Transaction Gateway\cicscli.log`	`logfile=<filename>` in the `CLIENT` section of the configuration file
	`Client daemon info log`	`<product_data_path>` `directory. By default, it is C:\ProgramData\IBM\CICS Transaction Gateway\cicscli.log`	`logfileinfo=<filename>` in the `CLIENT` section of the configuration file
	Gateway daemon error log	`<product_data_path>` `directory. By default, it is C:\ProgramData\IBM\CICS Transaction Gateway\cicstg.log`	`log@error.parameters=filename=<name>` in the `GATEWAY` section of the configuration file
	Gateway daemon info log	`<product_data_path>` `directory. By default, it is C:\ProgramData\IBM\CICS Transaction Gateway\cicstg.log`	`log@info.parameters=filename=<name>` in the `GATEWAY` section of the configuration file
On UNIX and Linux platforms	`Client daemon log`	`By default, it is /var/cicscli/cicscli.log`	`logfile=<filename>` in the `CLIENT` section
	`Client daemon info log`	`<product_data_path>` `directory. By default, it is /var/cicscli/cicscli.log`	`logfileinfo=<filename>` in the `CLIENT` section
	`Gateway daemon info log`	`console`	Specify in the `GATEWAY` section of the configuration file `log@error.dest=<file>` to have `log@error.parameters=filename=<name>` active
	Gateway daemon error log	`console`	Specify in the `GATEWAY` section of the configuration file `log@info.dest=<file>` to have `log@info.parameters=filename=<name>` active

Operating system	Log name	Default location	Override location
z/OS	`Gateway daemon log`	JES Joblog	`STDOUT` and `STDERR DD` statements in the Gateway daemon startup job

The Gateway daemon log handler allows you to specify the maximum log size also.

5.4.3 Gateway daemon memory requirements

To avoid out-of-memory conditions in the Gateway daemon, you need to carefully consider many elements. Most of the Gateway daemon resources are allocated in the Java heap and Java stack, and you can control them using the parameters **-Xmx** and **-Xss**.

The JVM maintains two memory areas: the Java heap, and the native (or system) heap. These two heaps have different purposes and are maintained by different mechanisms.

The Java heap contains the instances of Java objects and it is maintained by garbage collection. In the z/OS environment, the maximum size of the Java heap is preallocated during JVM startup as one contiguous area, even if the minimum heap size setting is lower. But in other platforms, the Java heap is incrementally acquired; for this reason, it must be carefully sized.

The Java heap is a repository for active, orphaned objects and free memory. When an object can no longer be reached from any object referenced in the running program, the object can be garbage-collected.

When the JVM cannot allocate an object from the heap because of lack of contiguous space, a memory allocation fault occurs, and the garbage collector is called. This process is a normal process invoked to tidy up the heap storage, but it consumes CPU time. For this reason, a meticulous evaluation of the heap storage is important. In effect, the heap size determines how often and how long the JVM spends performing the garbage collection.

The **-Xgcpolicy** options control the behavior of the garbage collector. For CICS Transaction Gateway, the best performance is provided by `optthruput`. The Gateway daemon default is also `optthruput`.

To tune the heap size, review the maximum and average `SE_CHEAPGCMIN` statistic value, through a workload cycle, as a percentage of `SE_SHEAPMAX` to ensure that the value is in the range of 40% - 70%. If the percentage is over 70%, increase the maximum heap size. If the percentage is under 40%, reduce the maximum heap size.

The system, or native heap, is allocated by using the malloc and free mechanisms of the operating system. Native heap is used for the underlying implementation of particular Java objects, such as the allocation of the Java threads.

5.5 z/OS specific resources

This section describes the z/OS specific resources and how to evaluate their correct usage.

5.5.1 Region size consideration in CICS TG for z/OS

Gateway daemon for z/OS memory management is complex. Several concepts are required to better tune the Gateway configuration.

An easy way to estimate the real amount of native heap required by the Gateway daemon at run time is to set the initial and maximum number of connection managers and worker threads to the same. This forces the Gateway daemon to create all the threads at startup and you can verify the actual storage consumption and act accordingly.

This suggestion will give you a reasonable sizing of the memory needed by your Gateway daemon. After that, you can start to tune the storage requirement more carefully.

Table 5-4 describes a base suggestion about how to evaluate the necessary storage allocation. This table represents a guideline, not an accurate formula, and the values can change depending on environmental factors, such as operating system release or Language Environment runtime options.

Table 5-4 Observed use of memory in a 31-bit Gateway daemon

Storage type	Usage
Native	356 KB per thread
Java heap	300 KB per IPIC session
Java heap	1.6 KB per thread
Java heap	4800 KB core Gateway daemon

The Java heap can be modified at the Gateway daemon startup by using the **-Xmx** option. The default heap size specified by the CICS Transaction Gateway is 128 MB, and this is typically increased for production configurations.

The JVM is a z/OS UNIX process that runs in its own Language Environment enclave. The JVM Heap, control blocks, and other data areas are handled by the Language Environment heap and it is governed by the Language Environment `HEAP` runtime option. Language Environment `stack` is allocated per thread with an initial size and can grow, according to the `ANYHEAP` runtime option values.

CTGBATCH runtime options, provided by default, are shown in Example 5-1, and you can override these parameters with the Language Environment runtime option specification at the Gateway daemon startup. Any change in the runtime option must be carefully evaluated.

Example 5-1 Default CTGBATCH runtime options

```
(ABTERMENC(ABEND),ALL31(ON),ERRCOUNT(0),
        STORAGE(NONE,NONE,NONE ,OK),
        TRAP(ON,SPIE),LIBSTACK(4K,4K,FREE),
        STACK(128K,128K,ANY,KEEP),
        ANYHEAP(16K,8K,ANY,FREE),
        BELOWHEAP(8K,4K,FREE),
        HEAP(32K,32K,ANY,KEEP,8K,4K),
        RPTOPTS(OFF),RPTSTG(OFF),POSIX(ON),
        TERMTHDACT(UADUMP),NOUSRHDLR)
```

Java heap can be modified at the Gateway daemon startup by using the **-Xmx** option. The default heap size specified by the CICS Transaction Gateway is 128 MB.

There are no options with which you can manage the native heap because this is done by the Gateway daemon for you. You only need to define enough region size to allocate the native heap requested.

JVM is constructed in its own Language Environment enclave, created by the Language Environment pre-init module. The JVM is a z/OS UNIX process. The JVM Heap, controlblocks, and other data areas are handled by Language Environment heap and it is governed by the Language Environment HEAP runtime option.

The JVM Heap and native storage used by the Gateway daemon is all managed by Language Environment.

The storage allocated by Language Environment is constrained by the Gateway daemon address space storage. For a 31-bit Gateway daemon, address space storage is set by the **REGION** parameter in the JCL (`EXEC` or `JOB` statement).

The following parameters can also limit the values that you can choose for the **REGION** parameter:

- The **ASSIZEMAX** parameter of the OMVS segment of a RACF user ID; for more information see *z/OS V1R13.0 Security Server RACF Command Language Reference*, SA22-7687-16:

  http://www-01.ibm.com/support/docview.wss?uid=pub1sa22768716.
- The UNIX System Services **MAXASSIZE** parameter is specified in `SYS1.PARMLIB(BPXPRMxx)`; for more information, see the UNIX System Services Planning. The value specified for **ASSIZEMAX** overrides any value provided by the **MAXASSIZE** parameter.
- If you use **REGION**=`0M`, the actual region size used is governed by the optional implementation of the system exit routine `IEFUSI`.

If you decide to set **REGION**=`0M`, it indicates that the address space must be given as much memory as possible, which can result in a system `abend878` or `abend80A`.

The local system queue area (LSQA) grows from the top to the bottom and REGION size grows from the bottom to the top. The free storage that is available is in the middle of the two. If there is no limit in the REGION size, it can grow until it reaches the LSQA limit, causing an abend.

`Abend878` and `abend80A` happen if there is not enough contiguous MVS storage available to satisfy an **MVS GETMAIN** request. This potential problem is illustrated in Figure 5-6 on page 84.

Address sample	Storage area	User	Usage
x80000000	**ELSQA** (1)	Core GW	193 MB
ELSQA bottom		Threads	83 KB / thread
	<<<<<<<<<<<Free Extended Storage>>>>>>>>>>>		
ETOP	**Extended user region**	Native	356 KB / thread
		Java heap	300 KB / IPIC session
			1.6 KB / thread
x1A000000			4800 KB Core GW
x10000000	**Extended Global**	MVS allocated (fixed at IPL and valid for all the address spaces)	160 MB
	<<<<<<<<<<<16 MB Line>>>>>>>>>>>		
	Global	MVS allocated (fixed at IPL and valid for all the address spaces)	8 MB
x00800000	**LSQA** (1)		
	<<<<<<<<<<<Free Storage>>>>>>>>>>>		
x00006000	**User region**		
x00000000	**System**		24 KB

Figure 5-6 Typical memory allocation for a 31-bit Gateway daemon address space on z/OS

Footnote 1 in Figure 5-6 shows that for LSQA/SWA/229/230", the size normally grows to a high-water-mark and then remains constant.

Note: `Java.lang.OutOfMemory` is not always a sign of insufficient heap. If there is an error in the thread creation, it manifests, in Java, as *out of memory*.

For more information about memory usage, see the topic, *Avoiding out of memory conditions for additional suggestions*, in the CICS TG Information Center:

http://ibm.co/Wjxls7

5.5.2 Storage and EXCI in the Gateway daemon for z/OS

The EXCI support in the Gateway influence the storage allocation needs.

When EXCI is enabled (by default), the following formula to calculate potential maximum virtual storage requirements of the Gateway daemons is in operation:

```
70000 + (heap max/ 1024) + (128 * maxconnect) + (1200 * maxworker)
```

When EXCI is disabled (environment variable **CTG_EXCI_INIT**=NO), the following formula is in operation:

```
70000 + (heap max/ 1024) + (128 * maxconnect) + (650 * maxworker)
```

Note: The `TRACESZE` defined in `DFHXCOPT` will be allocated as 16 Kb if the trace is `OFF`. Carefully consider the size of the trace because it will be allocated for each of the worker threads.

5.5.3 Using a 64-bit Gateway daemon on z/OS

If you run your Gateway with large numbers of clients and large container payloads (that require a heap size more than 2048 MB), consider using a 64-bit z/OS Gateway. This is possible implementing the 64-bit Java technology, using the 64-bit JDK.

Configuring the Gateway to run as a 64-bit z/OS Gateway, consider the z/OS JCL parameter **`MEMLIMIT`**. It sets the limit on how much virtual storage above the bar the Gateway can use. If you do not set **`MEMLIMIT`**, the system default is `0`, which means that no address space can use virtual storage above the bar. You can set an installation default **`MEMLIMIT`** through `SMFPRM`*`xx`* in `PARMLIB` or by using the `IEFUSI` exit.

You can also benefit from the **`-Xcompressedrefs`** Java option. It allows you to compress references. The JVM stores all references to objects, classes, threads, and monitors as 32-bit values. The **`-Xcompressedrefs`** and **`-Xnocompressedrefs`** command-line options enable or disable compressed references in a 64-bit JVM. Only 64-bit JVMs recognize these options.

A command-line option can be used with **`-Xcompressedrefs`** to allocate the heap specified with the **`-Xmx`** option, in a memory range of your choice. This option is **`-Xgc:preferredHeapBase=<`*`address`*`>`**, where ***`<address>`*** is the base memory address for the heap.

In Example 5-2, the heap storage of 2048 MB is located at the 4 GB mark. This leaves the lowest 4 GB of address space for use by other processes, and the references are kept in the 64-bit storage.

Example 5-2 Heap storage at 4 GB mark example

```
CTGSTART_OPTS=-j-Xms256M -j-Xmx2048M -j-Xgc:preferredHeapBase=0x100000000
-j-Xcompressedrefs
```

For a detailed explanation about tips for Java on z/OS, see this website:

http://www-03.ibm.com/systems/z/os/zos/tools/java/faq/javafaq.html

The 64-bit z/OS Gateway implementation also permits the use of z/OS *large pages*. The z/OS large page provides better performance by decreasing the number of translation lookaside buffer (TLB) misses that an application can incur. The large page support increases the page size from 4 KB to 1 MB. It was introduced in z/OS V1R10 for the IBM z10™ EC models. You need to check your hardware level and operating system level for the support of z/OS large pages. When large pages are used in addition to the existing 4 KB page size, they are expected to reduce memory management overhead for maximizing applications.

To implement these large pages, you need to specify the amount of real storage to be used for managing them. Management is done by coding the keyword `LFArea` in `IEASYS`*`xx`*, as shown in Example 5-3 on page 86.

Example 5-3 Coding for z/OS large page

```
LFArea=(xx%| xxxxxxM | xxxxxxG)
xx% - Percentage of online storage at IPL to be used for large pages
xxxxxxM or xxxxxxG – Amount of online storage at IPL to be used for large pages
```

Large pages are backed by 256 contiguous 4 KB real storage frames. Large pages will not be pageable. For this reason, you need to estimate the real storage usage.

With z/OS V1R13, a new command was introduced that lets you display the `LFAREA` current usage, as shown in Example 5-4.

Example 5-4 New command to display the amount of LFAREA

```
D VIRTSTOR,LFAREA

IAR019I 11.34.10 DISPLAY VIRTSTOR 846
SOURCE = 00
TOTAL LFAREA = 100M
LFAREA AVAILABLE = 100M
LFAREA ALLOCATED (1M) = 0M
LFAREA ALLOCATED (4K) = 0M
MAX LFAREA ALLOCATED (1M) = 12M
MAX LFAREA ALLOCATED (4K) = 0M
```

For an additional description of large pages, see *z/OS Version 1 Release 13 Implementation*, SG24-7946.

5.6 z/OS Workload Manager suggestions for CICS TG for z/OS

MVS Workload Manager provides a solution for managing workload distribution, workload balancing, and distributing resources to competing workloads.

Workload management allows you to make the best use of your resources, maintain the highest possible throughput, and achieve the best possible system responsiveness. With workload management, you define performance goals and assign a business importance to each goal.

This section will describe parameters that can influence the Gateway daemon performance.

5.6.1 WLM system-provided special service classes

WLM provides service classes for various system and subsystem address spaces; one of them is called `SYSSTC`. Started tasks in `SYSSTC` are assigned a high dispatching priority. If the default service class is left blank, these started tasks are assigned to `SYSSTC`. For this reason, CICS TG can benefit from this workload association when started as a procedure, without any change in WLM service class association.

The main point is that CICS TG must not be run in the `BATCH` service class. For maximized utilization of your resources, remember that the Gateway daemon is an important address space that provides workload to CICS, and it must not be considered as a batch job.

5.6.2 Storage Critical option

In a constant Gateway daemon workload, the WLM storage critical parameter has no effect. However, if Gateway daemons do not receive workload during the night, the storage can be stolen from the region. Then, when the workload flow starts again, it can cause a delay until the system gets back the stolen pages. You can assign long-term storage protection to critical address space by using the Storage Critical option of the WLM policy.

With this option, WLM protects an address space by restricting the storage donation to other address spaces. This option can be useful for an address space that needs to retain virtual pages in main storage during long periods of inactivity because it cannot afford paging delays when it becomes active again. Without this function, if main storage enters contention, a stealing algorithm steals the pages least referenced. With long-term storage protection assigned, this transaction address space loses main storage (by stealing) only from transactions of equal or greater importance that need the storage to meet performance goals. The storage of protected service classes is only taken when z/OS runs short on main storage.

5.6.3 MAXCPUTIME

If your Gateway daemon is running for a long time, and it is serving a large workload, you need to consider the z/OS UNIX parameter **MAXCPUTIME**. z/OS WLM can suspend the Gateway daemon address space if the maximum CPU time provisioned for an address space is exceeded.

MAXCPUTIME is a time limit (in seconds) limiting the CPU time that a process is allowed to use. It is applicable to all the processes that run as daemons (in UNIX System Services terms, a *daemon* is a process that is started in the background), and therefore, the Gateway daemon is also subject to these limitations.

You can set a system-wide **MAXCPUTIME** limit in the `BPXPRMxx` member, or you can set the limit for the Gateway daemon user ID. Use the RACF commands **ADDUSER** or **ALTUSER** to specify the **CPUTIMEMAX** limit on a per-process basis, for example:

```
ALTUSER userid OMVS(CPUTIMEMAX(nnnn))
```

Alternatively, specify the parameter **CPUTIMEMAX** in the OMVS segment for the RACF user ID of the Gateway daemon.

5.7 Statistics

CICS Transaction Gateway is a key part of your IT infrastructure. It is therefore essential to optimize your CICS TG system efficiency to eliminate resource bottlenecks and to improve productivity.

The statistics produced by CICS TG allow you to analyze the CICS TG status in the following areas:

- Ensure optimal performance of CICS TG.
- Use your hardware assets optimally.
- Focus on CICS TG resource utilization.

Statistics are available in all CICS TG products, and they can be displayed through **modify** commands on z/OS and through the **ctgadmin** command-line tool on other platforms. They are also available for use by external tools, such as the CICS TG plug-in for CICS Explorer and IBM Tivoli OMEGAMON XE for CICS.

CICS TG also provides a mechanism to record interval and end-of-day statistics to System Management Facility (SMF) on z/OS and to an XML file on other platforms. CICS TG SMF records can be analyzed using CICS Performance Analyzer, or by writing your own program to format the statistical records.

For detailed information about the statistics feature, see *Exploring Systems Monitoring for CICS Transaction Gateway V7.1 for z/OS*, SG24-7562.

6

CICS Transaction Gateway solution options

This chapter describes the connectivity between a client application and a CICS program through the CICS Transaction Gateway (CICS TG). It compares local mode and remote mode connections and outlines considerations for the connection options in relation to the client platform and APIs in use.

This chapter covers the following topics:

6.1 Deployment topologies

The CICS TG supports two types of client connections referred to as l*ocal mode* and *remote mode*. The mode that you use depends on your application topology. If the client application executes on the same server where the CICS TG is running, local mode is appropriate. If the client application executes on a separate server from where the CICS TG is running, only remote mode is supported. Figure 6-1 shows the local and remote mode configurations with CICS TG for Multiplatforms.

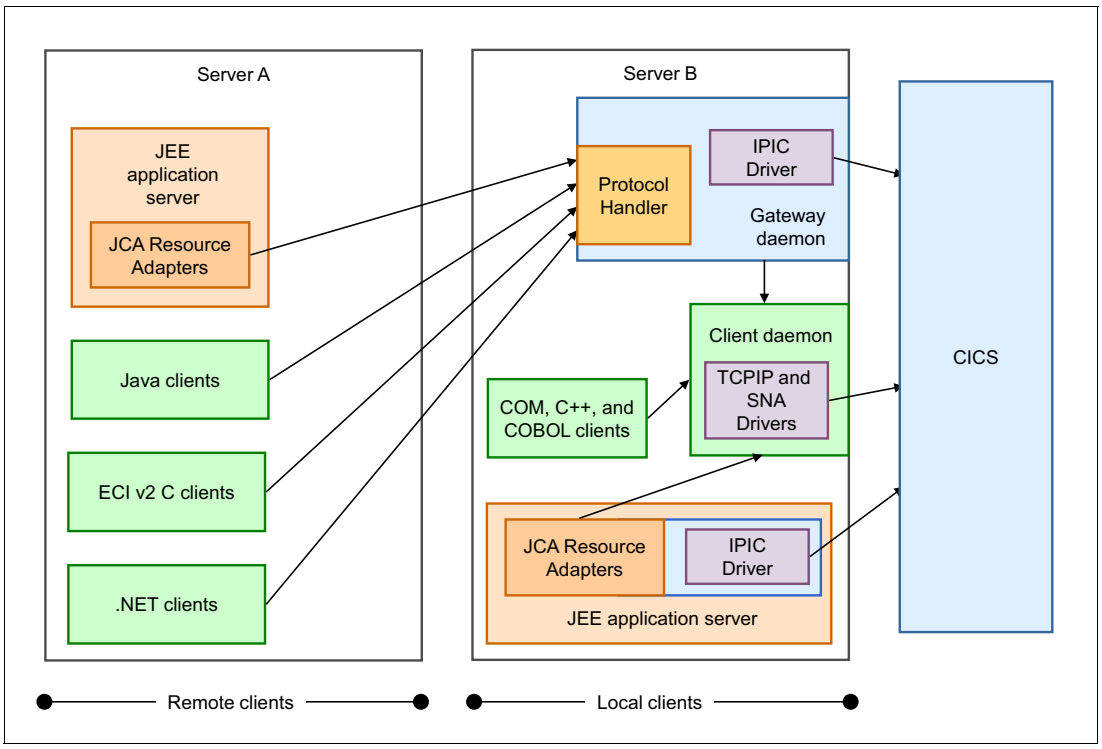


Figure 6-1 Local mode and remote mode clients with CICS TG for Multiplatforms

In Figure 6-1, applications running on Server A can connect to the Gateway daemon on Server B across the network. These are remote mode clients. C, C++, and COBOL programs using the original ECI, external presentation interface (EPI), and external security interface (ESI) are not capable of connecting across the network and therefore must execute on the same server where the CICS TG is installed. Their local mode connections use the Client daemon directly. Local mode Java applications executing on Server B use the CICS TG code within their own Java virtual machine (JVM) rather than connecting to the Gateway daemon. They can use the IPIC driver or connect to the Client daemon when using the TCP/IP and Systems Network Architecture (SNA) protocols.

Note: Remote mode client applications can be executing in single user systems and servers.

Local and remote mode clients can also be used with CICS TG for z/OS. Figure 6-2 shows the same capability for deploying remote client applications as the capability with CICS TG for Multiplatforms. However, the only local clients supported with CICS TG for z/OS are Java and JCA clients. Also, z/OS has no Client daemon. The Gateway daemon and local mode Java clients access the external CICS interface (EXCI) protocol through a Java Native Interface (JNI) call to the CICS-supplied EXCI module.

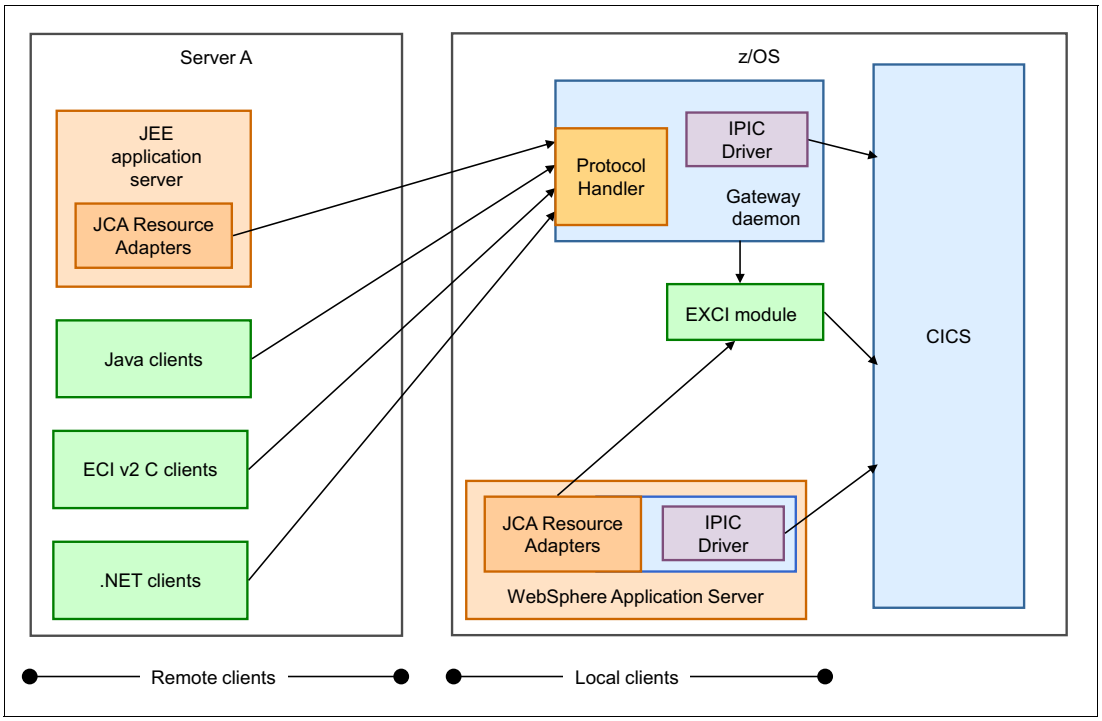


Figure 6-2 Local mode and remote mode clients with CICS TG for z/OS

CICS TG Desktop Edition is a special case when it comes to remote and local clients. The license limits its use to a single user, and it does not permit redistributing components. This means that remote mode client applications, such as Microsoft .NET Framework-based applications or those using the ECI version 2 (ECIv2), must be deployed into the same desktop system where CICS TG is installed. Figure 6-3 on page 92 illustrates these local client applications that use the remote mode configuration and standard local mode clients.

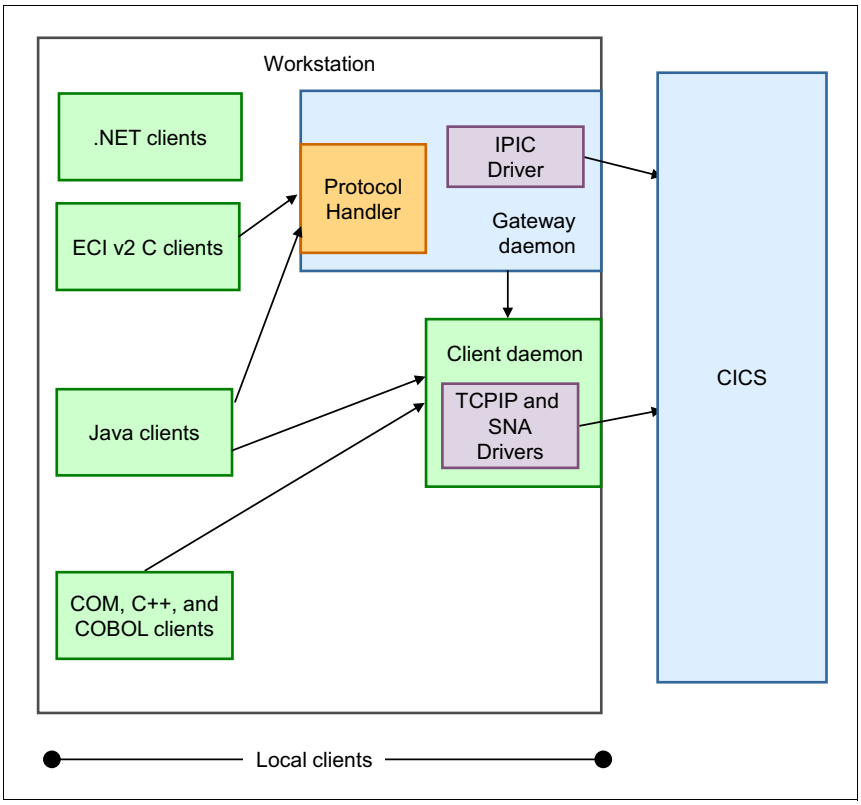


Figure 6-3 Client applications with CICS TG Desktop Edition

6.1.1 Remote mode

A remote mode configuration uses the Gateway daemon. The client establishes an IP connection to the Gateway daemon by specifying the IP address or host name of the server where the Gateway daemon is executing. The configuration of the CICS TG specifies on which port the Gateway daemon is listening and whether Secure Sockets Layer (SSL) is to be used.

The CICS TG supports the following client applications as remote clients:

- JCA client applications
- Java client applications
- ECIv2 C client applications
- .NET client applications

Remote client applications communicate with the Gateway daemon using the following protocols:

- Transmission Control Protocol (TCP)
- SSL

By default, the TCP port is bound to 2006 and the SSL port is bound to 8050. Both server authentication and client authentication mechanisms can be used with the SSL protocol.

The license for CICS TG Desktop Edition restricts the use of remote mode clients to programs that are executing on the same machine where CICS TG is installed. These locally installed remote mode client programs can be coded to use the Java, ECIv2, or .NET client interfaces provided by CICS TG.

6.1.2 Local mode

On distributed platforms, local non-Java CICS TG client applications communicate directly with the Client daemon. Local Java (and JCA) client applications do not require the Gateway daemon but use the Gateway classes provided by CICS TG to communicate with the CICS system.

CICS TG for z/OS only supports Java and JCA clients in local mode. CICS TG for Multiplatforms supports the following client application types as local clients:

- C and COBOL client applications
- C++ client applications
- COM client applications
- JCA client applications
- Java client applications

CICS TG Desktop Edition supports all of the previously listed client types except for JCA.

On z/OS, the client application uses the Internet Protocol interconnectivity (IPIC) or EXCI protocols. For distributed platforms, the client uses the IPIC driver or connects directly to the Client daemon.

For a local mode configuration, there is no need to start the Gateway daemon. Note that on Windows platforms, the CICS TG runs as a Windows service which automatically starts the Gateway daemon.

Local mode client applications must set the ECI request calls to `local:`, under the **`URL`** parameter. When using the IPIC protocol in a local mode configuration, you do not need to configure the server connection definition in the configuration file. The connection definition is included in the server name parameter on the ECI request call or in the Java EE Connector architecture connection factory.

Note: Java and JCA client applications are supported in local mode if the client application and CICS TG are on the same server. However, they will not be able to use the features provided by the Gateway daemon.

6.2 CICS TG APIs and connection type

Each CICS TG product consists of API runtime components providing services to CICS TG applications, and CICS connectivity components for communicating with CICS servers. The CICS connectivity components are included in the product installation, and are licensed to run on the machine where the product is installed. The API runtime components are also included in the product installation, and several of these can be freely deployed to any number of Multiplatform machines for remote mode (3-tier) operation.

Not all API services are available in each product. The choice of programming API can impose limitations as to which product and connection type can be used. See Table 6-1 on page 94.

Table 6-1 Product and connection types based on the API Programming chosen

API component	Application run time	CICS TG for z/OS	CICS TG for Multiplatforms	CICS TG Desktop Edition
Java base classes	Java SE	Remote Local	Remote Local	Local[3,4]
JCA resource adapters	Java Platform, Enterprise Edition (Java EE) application server	Remote Local[1]	Remote Local[2]	N/A
ECIv2 and ESIv2	Multiplatforms C/C++ Multiplatforms COBOL	Remote	Remote Local[3]	Local[3,4]
.NET ECI and ESI	.NET Framework	Remote	Remote Local[3]	Local[3,4]
ECI, ESI, and EPI v1	Multiplatforms C/C++ Multiplatforms COBOL COM	N/A	Local[5]	Local[5]

1. Available for local or remote mode JCA connection factories, EXCI, or IPIC.
2. Available for local mode JCA connection factories, IPIC only.
3. Colocated application and Gateway daemon connect through the local network adapter.
4. Application, runtime component, and product must be colocated due to license restrictions.
5. Uses cross-memory service between application and Client daemon.

6.2.1 JCA client applications

JCA client applications are supported in both local and remote mode.

JCA connects enterprise information systems, such as CICS, to the Java EE platform. JCA supports the quality of service (QoS) for security credential management, connection pooling, and transaction management provided by a Java EE application server. QoS is provided through system-level contracts between a resource adapter provided by CICS TG and the Java EE application server.

CICS TG supplies two JCA resource adapters to be used by client applications. These resource adapters provide high-level common client interfaces (CCI) to the ECI and EPI for sending requests to CICS. The resource adapter can be deployed into a Java EE application server to allow Java EE enterprise applications to access CICS. When JCA is used in a Java EE application server, connection pooling, security, and transaction context are managed by the Java EE application server, not by the application.

Installation of the JCA resource adapter into the Java EE application server provides the basis for defining managed connection factories. In a Java EE application server, you create the connection factory by configuring it using an administrative interface, such as the WebSphere Application Server administrative console. You set custom properties, such as security parameters, transactional properties, and the Gateway connection URL (either a remote connection to the Gateway daemon or a local connection) for the connection factory. This type of environment is called a *managed environment*, and allows the Java EE application server to manage the QoS for each of the connections.

For JCA applications, which are not running within a Java EE application server, the application is responsible for creating the connection factory and setting its custom properties. This type of environment is called a *non-managed environment* and offers no QoS for the connections.

Restrictions

The following list shows important restrictions:

- Only ECI and EPI are supported by the JCA resource adapters.
- Automatic Transaction Initiation (ATI) is not supported by the EPI resource adapter.
- With CICS TG for Multiplatforms, XAResource transaction management is only available when using a local mode IPIC connection.
- The TCPIP or SNA network connections from the Client daemon into the CICS region are managed and reused by the Client daemon component of the CICS TG. They are not subject to the JCA connection pooling system contract.

6.2.2 Base class Java applications

CICS Transaction Gateway enables Java client applications to communicate with programs on a CICS server by providing base classes for ECI and external security interface (ESI), and external presentation interface (EPI) support classes.

A Java client application must create and manage its own connection to the CICS server before it can send a request. The application creates a logical connection between the Gateway daemon and itself when the application is running in remote mode. If a Java client application is running in local mode, the application creates a connection between the CICS server and itself, bypassing the Gateway daemon.

Restrictions

The following list shows important restrictions:

- When connecting to a remote CICS Transaction Gateway, resources allocated to a particular connection and identifiers specified on the request objects on a particular connection are available only to that connection.
- If you specify the `local:` protocol, all Java Gateways that are created in the same JVM, that is, the same process, have access to each other's allocated resources or specified identifiers.
- The Java base classes do not supply any provision for connection pooling. Standard connection management by the application (**`create`**, **`open, use`**, **`close`**, and **`destroy`**) can result in excessive overhead. The application can benefit from a user-created connection pool manager.
- ESI is only supported on IPIC and SNA connections.
- ESI is only supported when connecting to CICS servers on z/OS.
- EPI can only be used on SNA connections to all CICS servers and on TCP/IP connections to TXSeries.

6.2.3 Microsoft .NET framework-based applications

The .NET classes are supported on all Windows platforms. They can be used to build 32-bit and 64-bit remote mode applications.

The `GatewayConnection` class represents a connection to a Gateway daemon. The connection is opened in the constructor and remains open until the application explicitly closes the connection.

The .NET Framework offers a number of advantages when developing remote client applications:

- A consistent model, provided by the .NET class library, for all supported programming languages.
- High levels of security for applications used in remote mode topologies; method-level security using industry standard security technologies can be explicitly defined.
- Separation of application logic from presentation logic for easier maintenance and upgrade.
- Simplified debugging plus the availability of runtime diagnostics.
- Simpler application deployment.

Restrictions

The following list shows the restrictions for .NET Framework:

- The .NET classes are not supported in local mode.
- The .NET classes do not support EPI applications.
- ESI applications can only be used on IPIC and SNA connections.
- ESI applications are only supported when connecting to CICS Transaction Server for z/OS (CICS TS).
- There is no provided connection pooling facility. Standard connection management by the application (**`create`**, **`open`**, **`use`**, **`close`**, and **`destroy`**) can result in excessive overhead. The application can benefit from a user-created connection pool manager.

6.2.4 ECIv2 and ESIv2 applications

The ECIv2 API was introduced in the CICS TG V7.2 release. It enables a client application to communicate with a CICS TG running on a remote machine in much the same manner as the Java APIs. This function is similar to the function that is provided by the existing Client APIs in the CICS TG. However, an application written using ECIv2 is not required to run on the same machine as a CICS TG installation. Originally, ECIv2 only supported COMMAREA-based applications. CICS TG V8.0 introduced enhanced capabilities for ECIv2 clients by adding support for channels and containers.

In CICS TG V8.1, security capabilities of the CICS TG and its client applications were enhanced with the addition of support for password management for remote C applications using the ESIv2 API.

Restrictions

The following list shows the restrictions for ECIv2 and ESIv2 applications:

- As mentioned for the Microsoft .NET framework-based applications and base Java classes, there is no provided facility for connection pooling. Applications that are designed to connect to the Gateway daemon, flow one or two requests, and then close the connection can experience performance problems under a heavy load. The application can benefit from a user-created connection pool manager.
- ECIv2 and ESIv2 applications are only supported in remote mode deployments.

6.2.5 ECI, ESI, and EPI v1 applications

These original programming APIs are only supported on CICS TG for Multiplatforms and CICS TG Desktop Edition. They require the Client daemon for communication to the CICS server.

In addition to the callable interfaces for C and COBOL programs, these restrictions also apply to the class libraries for C++ and COM applications.

Restrictions

The following list shows the restrictions for ECI, ESI, and EPI v1 applications:

- No support for remote mode connections. These applications must execute on the same system where CICS TG is installed.
- IPIC connections are not supported.
- Channels and containers are not supported. Only COMMAREA applications can be accessed by ECI client applications.
- ESI requests are only supported on SNA connections.
- COM applications are restricted to ECI and EPI. ESI is not supported.

6.3 Transactional models

Within networked transactional systems, each system is either referred to as a *resource manager* or a *transaction manager*. The transaction manager controls the outcome of the transaction and is responsible for the recovery of its own resources. It must also be able to coordinate multiple resource managers. The resource managers control access to recoverable resources, and have to implement the necessary network flows and logging procedures to provide transactional coordination.

The CICS TG is capable of extending a transactional context into CICS from the client application. The transaction type can be an extended unit of work (equivalent to a Java local transaction) or two-phase commit (Java global transaction or XAResource). Additionally, the ECI request can flow to CICS as a non-transactional or SYNCONRETURN request.

CICS Transaction Gateway supports all three transactional models on EXCI and IPIC connections. The TCP/IP and SNA protocols support non-transactional and extended units of work.

CICS transactions are work that is initiated in a CICS region and that runs as a CICS task under a four-character transaction ID (`tranid`). At task initiation, CICS implicitly starts a unit-of-work (UOW) for all CICS transactions. This is usually the initial boundary of the transactional work to be undertaken. All updates to recoverable resources or requests to other transactional systems are now part of this UOW, until either a synchronization point (sync point) is reached within the CICS program or the CICS transaction finishes and the task terminates as illustrated in Figure 6-4 on page 98.

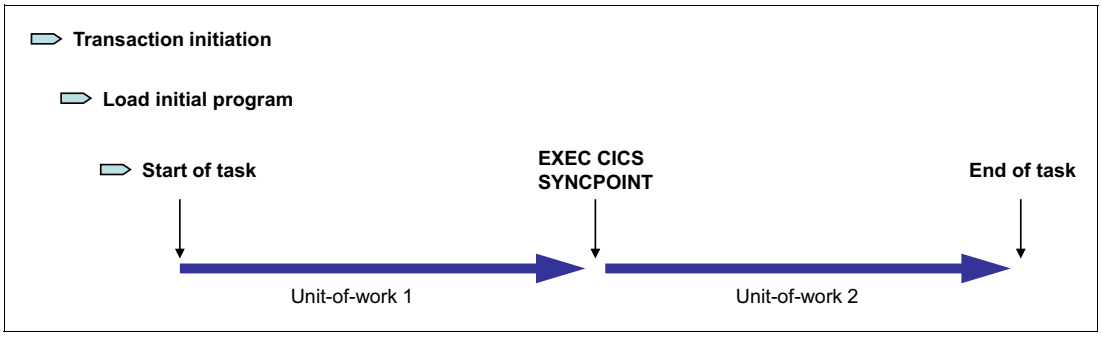


Figure 6-4 CICS synchronization points

When an inter-system distributed program link (DPL) request is used, the CICS transaction that is linked to can be coordinated by the remote system. In this instance, the mirror task remains suspended until the end of the transaction, which is referred to as a *long-running mirror task* as illustrated in Figure 6-5.

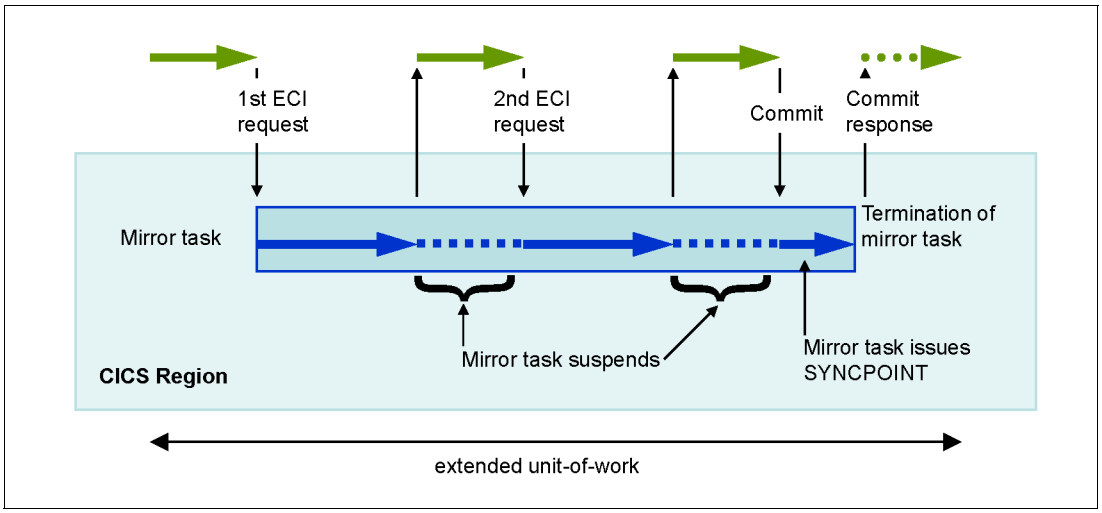


Figure 6-5 Link with long-running mirror task

Long-running mirror tasks are used to support both extended UOW and XAResource transactions. The CICS task is entered in the same way as a non-SYNCONRETURN DPL request and the programs in the CICS task are not allowed to issue commands that affect the state of the UOW. The CICS task's UOW is subordinate to the client application or the Java EE application server.

6.3.1 Two-phase commit

Two-phase commit is a designed set of flows that transaction managers use to ensure all resource managers in a transaction can be reliably coordinated, regardless of any failure.

Figure 6-6 on page 99 summarizes the flows according to the XA specification. Other protocols, such as CICS LU6.2, might use other terminology and variants on the flows. For further details about CICS sync point flows, see the CICS Transaction Server Information Center topic, *Syncpoint flows*:

http://ibm.co/1p37zCh

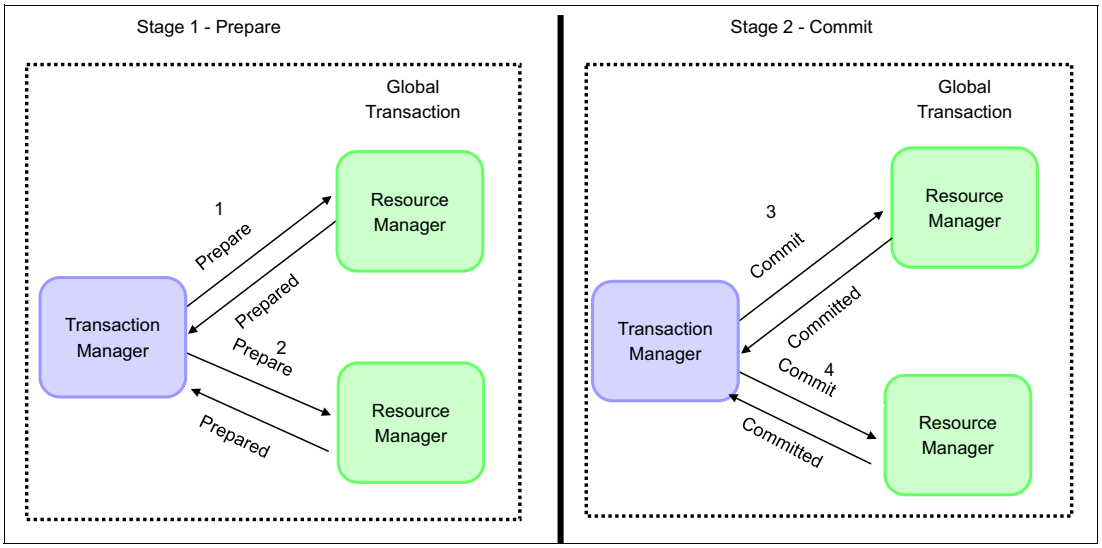


Figure 6-6 Two-phase commit

In the first phase (or stage 1), the transaction manager asks all the resource managers to prepare to commit recoverable resources. Each resource manager can vote either positively (prepared) or negatively (failed to prepare). If a resource manager replies positively, it is then obliged to follow the eventual outcome of the transaction as determined at the next stage. The resource manager is now described as in-doubt, because it has delegated eventual transaction control to the transaction manager. If a resource manager replies negatively, the transaction manager assumes that a backout is needed and the transaction will be backed out by all resource managers.

In stage 2, providing all the resource managers voted positively, the transaction manager replies to each resource manager with a commit flow. Upon receipt of the commit flow, the resource manager finalizes updates to recoverable resources, and releases any locks held on the resources. The resource manager then responds with a final committed flow, which indicates to the transaction manager that it is no longer in-doubt.

Although the two-phase commit process is usually a prerequisite to distributed transactional support, there are certain instances where a single-phase commit process can be sufficient. This is referred to as *last resource optimization*, and it is implemented by a variety of transaction managers. It essentially allows the commit decision to be delegated to the one-phase commit resource, allowing the one-phase commit to participate in a global transaction with any number of two-phase commit capable resources, as illustrated in Figure 6-7 on page 100.

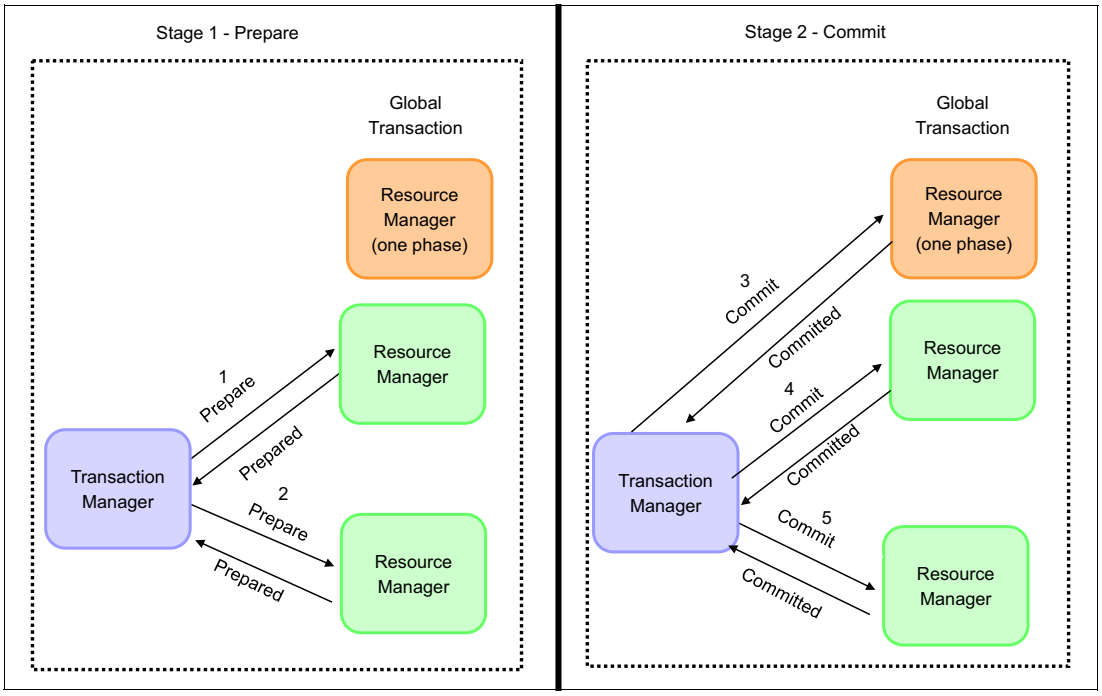


Figure 6-7 Two-phase commit with last resource optimization

At transaction commit, the transaction manager first prepares the two-phase commit resource managers, and if this is successful, the one-phase commit-resource is then called to commit. The two-phase commit resources are then committed or rolled back depending on the response of the one-phase commit resource, effectively delegating transaction coordination to the one-phase commit resource.

Unlike with a two-phase commit resource, there is no recovery from a communication failure with a one-phase commit resource. Such a communication failure during commit of the one-phase commit resource introduces the risk of a mixed outcome to the transaction. The two-phase commit resources are by default rolled back, but the outcome of the one-phase commit resource is unknown; it might have committed or rolled back. Applications must therefore be configured to accept the additional risk of such heuristic outcomes.

Last resource optimization is implemented within WebSphere Application Server as last participant support, and within CICS and Advanced Program-to-Program Communication (APPC) flows as last agent optimization. However, applications that are deployed to exploit last participant support are subject to an increased risk of a mixed outcome in a global transaction if the one-phase commit resource fails during the commit processing.

6.3.2 JCA and transactions

The JCA specifies the system contract for transaction management (and other system contracts such as connection management and security management) that exists between the Java EE application server and the enterprise information system (in this scenario, CICS).

A resource adapter is required to implement one of the following listed transaction management contracts, as defined in the resource adapter's deployment descriptor (`ra.xml`):

XATransaction A resource adapter that can participate fully in a two-phase commit process and can influence the outcome of the global transaction.

LocalTransaction A resource adapter that can participate in transactions that are local to the resource manager but that cannot participate in two-phase commit transactions (other than as an agent only or a last participant).

NoTransaction A resource adapter with no transactional properties that can participate in a transactional context but is not influenced by and has no effect on the outcome of the transaction.

CICS TG includes a CICS ECI resource adapter, which provides support for both LocalTransaction and XATransaction. The resource adapter provides LocalTransaction support when deployed on any supported Java EE application server. It can also provide XATransaction support when deployed with the custom property `xasupport=on` in any supported Java EE application server, as noted in the following list:

- Connecting to a remote CICS TG for z/OS
- When configured in a local mode with CICS TG for Multiplatforms

It also provides global transaction support when using WebSphere Application Server for z/OS with CICS TG on z/OS in local mode.

When the CICS ECI resource adapter is deployed in WebSphere Application Server with the custom property `xasupport=off`, the application server uses its last participant support when including CICS resources with other resources in a common global transaction.

Restrictions

The following restrictions apply to transactional support:

- CICS TG for Multiplatforms can only deliver support for XAResource coordination when configured in local mode within a Java EE application server.
- Local transactions are not supported when using WebSphere Application Server for z/OS with CICS TG for z/OS in local mode, because the resource adapter provides global transaction support with MVS Resource Recovery Services (RRS).

6.4 Choosing a Gateway topology

Depending on the programming API that you select, you might not have a choice of local versus remote topology. That is, ECIv2 and Microsoft .NET Framework-based applications are only implemented in remote mode. COM and C++ applications are only supported in local mode with CICS TG for Multiplatforms and CICS Desktop Edition. The original ECI, ESI, and EPI v1 interfaces are only available with CICS TG for Multiplatforms and CICS Desktop Edition and can only be configured in local mode.

When designing the topology for client applications, CICS Transaction Gateway, and CICS servers, you will need to be aware of the restrictions listed under each of these programming APIs. The design of your application can make it necessary to implement a specific configuration to meet its functional requirements.

Similarly, your choice of advanced capabilities can restrict your choice of deployment options. When using CICS TG for Multiplatforms, XATransaction support is only available when executing in a supported Java EE application server configured in a local mode.

A local mode configuration is best suited to smaller scale systems. A local mode configuration has the following features:

- Fewer components to manage than in remote mode
- Simplified network topology

Remote mode provides the following additional Gateway daemon capabilities:

- Gateway daemon statistics and monitoring facilities
- Network concentration for multiple remote clients
- High availability support

7

High availability concepts

This chapter introduces high availability concepts to build a foundation for Chapter 13, “High availability configuration” on page 295, where a sample scenario is described.

This chapter describes the available options to avoid a single point of failure (SPOF) and how to enable a system for scaling.

This chapter covers the following topics:

7.1 High availability overview

A *high availability system* is a system that delivers an agreed-on level of service during scheduled periods of availability. To achieve a high availability system, you have to eliminate any unscheduled unavailability that can occur. Practically, high availability systems strive for at least 99.9% availability during the scheduled time.

Continuous operation is the theoretical operation of a system for 24 hours per day, 365 days per year. In a continuous operation system, you make no provision for scheduled unavailability. All changes and maintenance must be applied in a nondisruptive manner.

Continuous availability combines the characteristics of both continuous operation and high availability. No unavailability of any nature is allowed.

For the purpose of this book, *high availability* (HA) is the ability of a CICS Transaction Gateway (CICS TG) to continue to provide its services regardless of component failures or planned outages.

To fully understand which role CICS TG is playing in a continuous availability system, and which is the best strategy provided by CICS TG to avoid unavailability, you need to identify the single points of failure (SPOF).

The SPOF is a part of a system, which, if it fails, will prevent the system from working fully. SPOFs are undesirable in any system with a goal of high availability. Duplication and redundancy are the basic principles used to eliminate any SPOF.

Another important consideration is the possible cause of unavailability. When you create a redundant infrastructure to avoid a SPOF, you need to allow the user to interact with a single entity, even if more than one system is up and running to satisfy requests. This is called a *single system image*.

To have a well-balanced single system image, you can also consider *dynamic workload balancing* and the possibility of *incremental growth*. The workload must be equally distributed through the systems to guarantee a quality of service (QoS), so the high availability system needs to be flexible for scaling.

CICS TG has to answer to these questions:

- What does HA mean?

  CICS TG HA is used to make CICS regions highly available to remote clients that access CICS using the facilities of the CICS TG remote APIs. Applications will appear continuously available to the users.

- When is the CICS TG HA needed?

  It is used to meet service level agreements (SLA) and other obligations.

- Why is the CICS TG HA needed?

  It allows at least one Gateway daemon instance to remain active during planned maintenance activities. It allows the restart of the Gateway daemon if new resources must be installed. It minimizes the impact of an outage.

- How can CICS TG HA be implemented?

  By using the Sysplex Distributor capabilities and TCP/IP port sharing to provide a single system image, cloned Gateway daemon instances eliminate points of failure, provide a workload balancing mechanism, and eliminate or reduce the impact of any affinities.

  CICS TG offers the capability to modify the CICS server specified by the application, or to provide one when it is not specified. This is possible using the Dynamic Server Selection (DSS) facilities, such as the default server setting, CICS Request Exit, or on z/OS, `DSSPOLICY`.

7.2 Connection balancing

This section highlights the SPOF exposure of various solutions from the client perspective considering only the link between the client and the Gateway daemon. Then, it describes the CICS TG answer to the exposures.

When the users connect to the application through the network, they are not interested in the design of the network and its complexity. If the application runs and responds as expected, it is not important to know how many CICS TG and CICS Transaction Server (CICS TS) instances are managing the request. The user will see only a *single system image system*. If the SLA was not affected, it is not important for the user to know, for example, whether a particular Gateway daemon was shut down and whether a peer Gateway daemon processed the requests instead.

If the infrastructure is hidden to the user, it is also possible to scale it transparently if the workload increases. A Gateway daemon can share the same IP port with another CICS TG instance, allowing easy scaling of the system, but it also appears as a single system image to the user.

7.2.1 Single point of failure: Single Gateway daemon

If the client attempts to connect to a specific Gateway daemon, and that Gateway daemon is not available, the request cannot be satisfied. Figure 7-1 shows the impact of having a single Gateway daemon solution; it is a SPOF in the overall solution, and access to the CICS systems might not be possible.

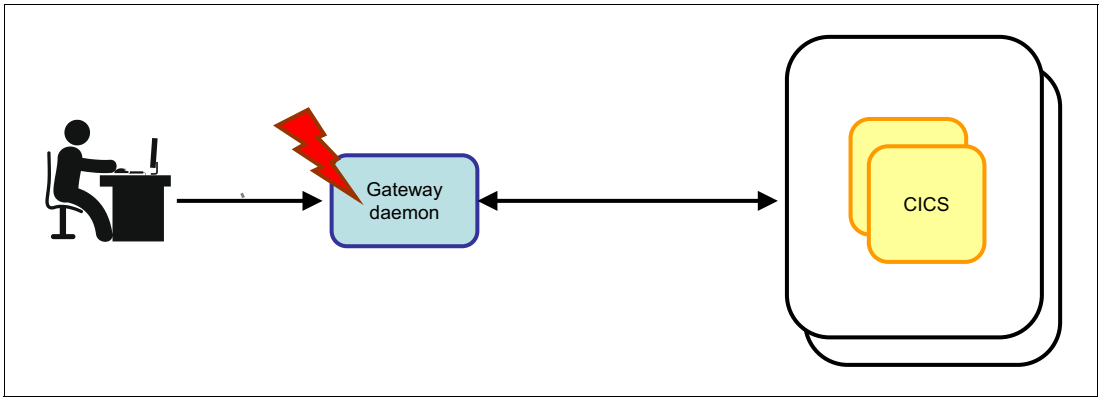


Figure 7-1 The single point of failure of a single Gateway solution

To resolve this issue, you can add another Gateway daemon instance to the solution to act as a substitute when the primary Gateway daemon is not available. Figure 7-2 shows a possible solution to the issue.

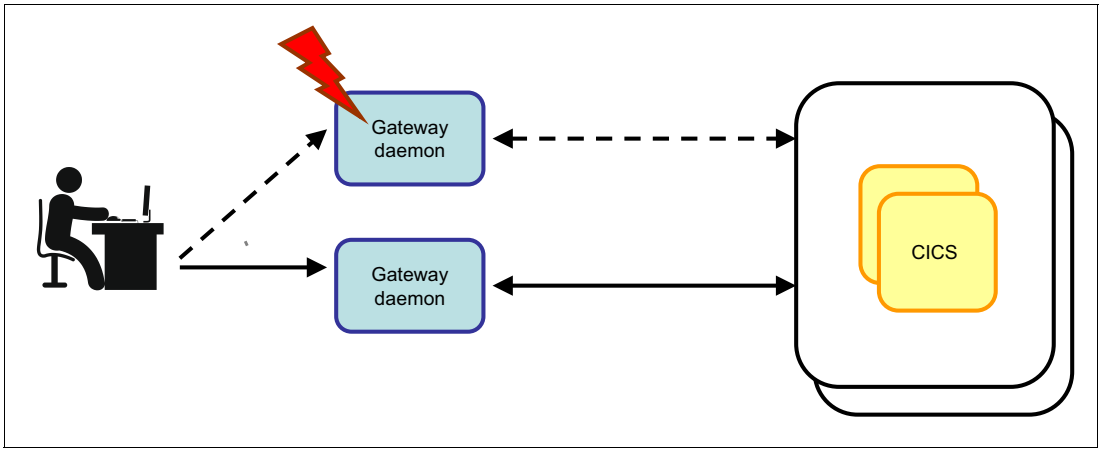


Figure 7-2 Multiple Gateway daemon solution removes the SPOF

In TCP/IP networking, Internet Protocol (IP) addresses are typically assigned to physical network interfaces. This point illustrates another issue: to use the substitute Gateway daemon, the client must switch the connection from the address of the unavailable Gateway daemon to the address of the running one. To avoid manual intervention, the shared port solution provides an answer.

To guarantee the single system image, the ability to scale, and to avoid the SPOF, you need to share the TCP/IP port between the different Gateways, so the client can address a single port, and the infrastructure will be transparent to the user.

Figure 7-3 shows that the clients see only TCP/IP port 2006, and both Gateways are addressed.

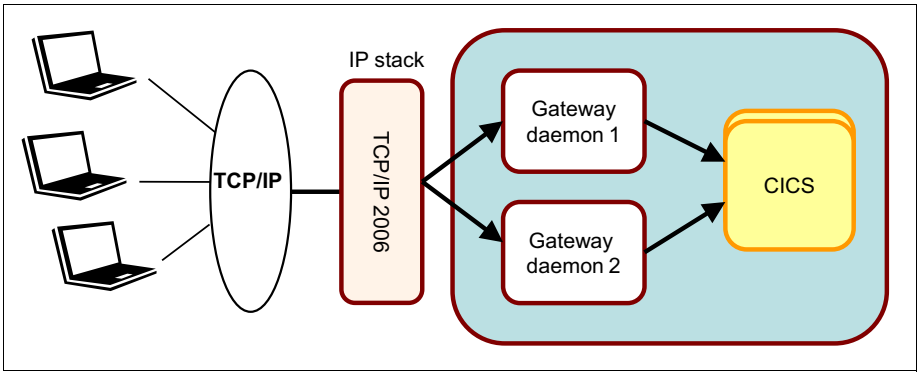


Figure 7-3 Implementation of shared ports

7.2.2 z/OS shared ports

This section describes the capabilities offered by z/OS to share the TCP/IP ports between multiple Gateway daemons. It is useful to introduce terminology to assist the description:

- *Stack* is commonly used to refer to the TCP/IP address space.
- A virtual IP address (*VIPA*) allows the use of IP addresses, representing TCP/IP stacks, applications, or clusters of applications, that are not tied to any specific physical interface. There is an association between a VIPA and an actual physical interface.
- A VIPA definition can be *static*, where it is typically associated with a particular TCP/IP stack, or dynamic, which is known as a *DVIPA*. A DVIPA is defined by a `VIPADYNAMIC` statement in the TCPIP profile on multiple stacks, and it can be automatically moved between stacks within a Parallel Sysplex. The DVIPA definitions used with Gateway daemons are also known as *multiple application-instance DVIPA*.
- A VIPADYNAMIC definition can include the `VIPADISTRIBUTE` statement to define a *distributed DVIPA*. This is used to designate stacks and ports where connections for a particular DVIPA can be distributed.
- An *advertised DVIPA* is known to external systems, or client applications. Typically, work arriving to this external address is distributed to specific stacks on logical partitions (LPARs) within the Parallel Sysplex.

For further detail about the DVIPA topology, see the topic, *Multiple application-instance DVIPA*, in the Networking on z/OS Information Center:

http://ibm.co/1p37Gh8

For further explanation of terminology related to virtual IP addressing, see the topic, *Static VIPAs, dynamic VIPAs, distributed DVIPAs*, in the *z/OS Communications Server: IP Configuration Guide:*

http://ibm.co/UgHcxM

TCP/IP port sharing and the Sysplex Distributor

The z/OS Communications Server IP (also known as TCP/IP for z/OS) provides a function called *TCP/IP port sharing*.

This technology allows you to have multiple Gateway daemons, on the same logical partition (LPAR), that listen on the same port. If one Gateway daemon is unavailable, the other instances process the workload transparently for the user.

TCP/IP port sharing

TCP/IP port sharing also provides connection balancing via round-robin or Workload Manager (WLM) distribution, based on the weight fraction, normalized value.

For TCP/IP port sharing, the parameters **`SHAREPORTWLM`** and **`SHAREPORT`** influence the routing:

- When **`SHAREPORT`** is specified on the `PORT` statement, incoming client connections are distributed by the stack across the listeners, using a weighted round-robin distribution method based on their Server Efficiency Fraction (SEF) values.

  SEF measures how well a target server is accepting new TCP connection requests and managing its backlog queue for socket accept operations.

- When **SHAREPORTWLM** is specified on the PORT statement, incoming client connections are distributed by the stack across the listeners, using WLM recommendations based on the Target Server Responsiveness (TSR) values for each listener:
  - Target Connectivity Success Rate (TCSR) is the measure of how many new TCP connection requests are received by the target stack.
  - TSR is calculated by combining two factors, the values for TCSR and SEF.

Note: SEF values are shown using the TCPIP display command, from TSO, provided that shared ports are configured:

D TCPIP,<*TCPIP job name*>,NETSTAT,VDPT,DETAIL

TSR is evaluated at intervals of approximately one minute. If a target server experiences response problems, new connection requests are diverted away from that server.

In this solution, there is still a SPOF, which is the LPAR. If the LPAR fails, the user will not be able to work. To solve this problem, you can use the z/OS Sysplex Distributor.

Sysplex Distributor

z/OS Parallel Sysplex combines parallel processing with data sharing across multiple systems to enable your application to benefit from harnessing the power of plural z/OS mainframe systems. Yet, it makes these systems behave as a single, logical computing facility. This combination gives the z/OS Parallel Sysplex unique availability and scalability capabilities. The Sysplex Distributor provides TCP/IP connection balancing between listener ports of equivalent services, in different stacks.

Note: Port sharing can be used in combination with Sysplex Distributor.

Figure 7-4 shows an example of a Sysplex Distributor and TCP/IP implementation.

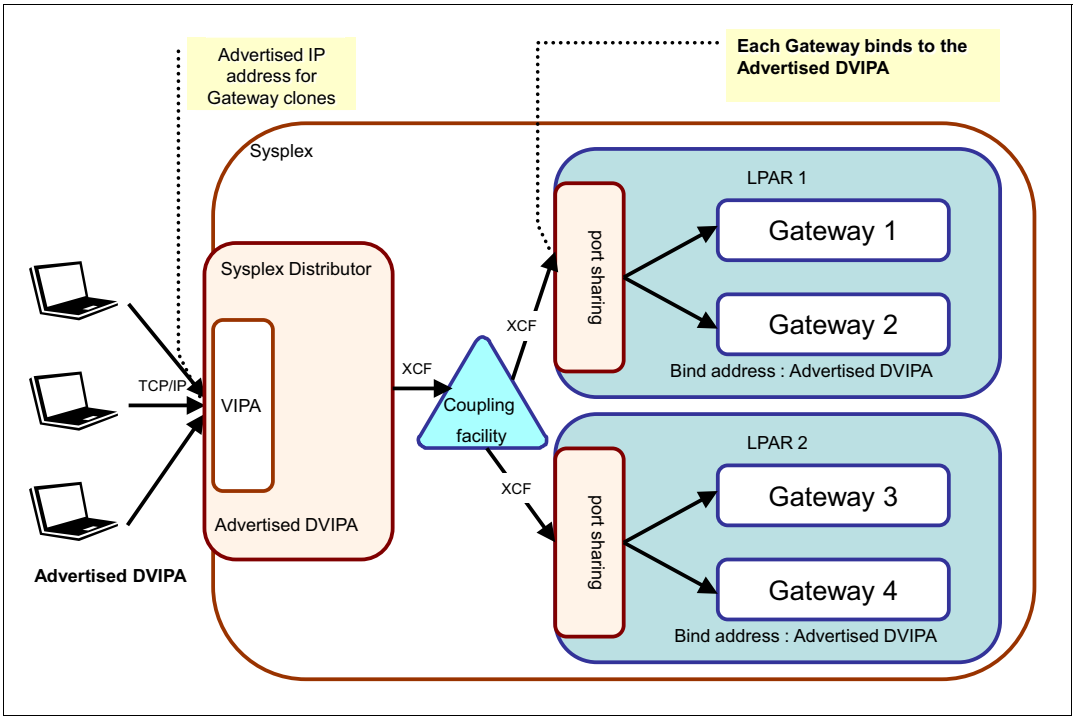


Figure 7-4 Example of z/OS shared ports and Sysplex Distributor

The Gateway daemon protocol handler listener ports (TCP and SSL) can be bound to the static VIPA addresses of their respective LPARs (in this book, 192.168.0.1 and 192.168.0.2). However, this can be regarded as a security exposure, because it allows connections to be established through both the advertised DVIPA (in this book, 1.2.3.4), and the static VIPAs. Binding to the advertised DVIPA limits the means by which an application can connect, providing greater control to the system administrator.

Figure 7-5 shows the main configuration steps needed for the implementation illustrated in Figure 7-4 on page 108.

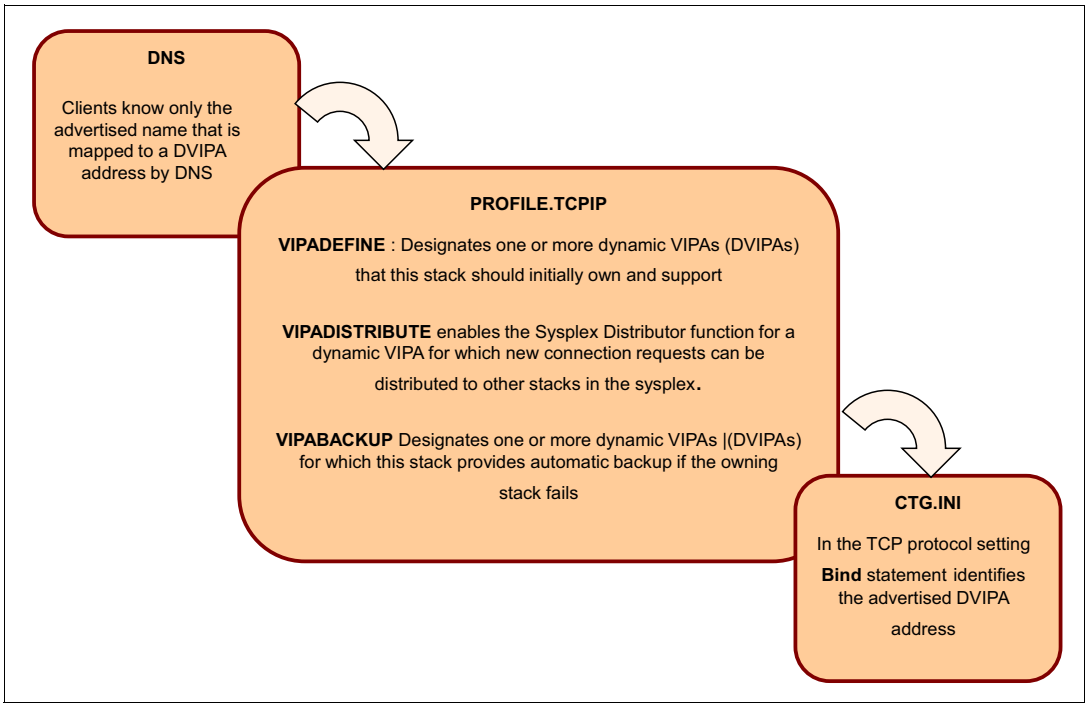


Figure 7-5 Sysplex Distributor and TCP/IP port sharing implementation

For a complete description of these features, see *IBM z/OS V1R12 Communications Server TCP/IP Implementation: Volume 3 High Availability, Scalability, and Performance*, SG24-7898.

Sysplex Distributor and TCP/IP port sharing make use of Workload Manager (WLM) and its ability to gauge server load and provide a WLM recommendation.

For Sysplex Distributor, when the distribution method for a target server is BASEWLM or SERVERWLM, the weights to the target server are modified to reflect its ability to handle new connection requests. Target servers whose weights are lowered because of this are less favored by the distribution algorithm:

- *BASEWLM* provides capacity recommendations in the form of weights for each *system*. When determining a WLM system weight recommendation, WLM assigns a relative weight to each system in the sysplex, with the highest weight going to the system with the most available capacity. The available capacity is based on the system's available general CPU capacity. BASEWLM uses the total available system capacity as the base for the weight.

- *SERVERWLM* provides capacity recommendations based on the importance of the workload that the server is running. When determining a WLM server-specific recommendation, WLM determines the service class of the server's address space. Then, WLM assigns a weight based on how well that server is meeting the goals of its service class, and the amount of displaceable workload available, given the importance of the service class.

The value of the SEF and the policy selected for balancing the TCP/IP causes incoming connections to be distributed among a set of TCP/IP listeners. Selection is based on WLM server-specific recommendations, modified by the SEF values for each listener.

How to handle failback for restarted Gateway daemons: If IP connection balancing is in used across a group of cloned Gateway daemons, restarted Gateways might not be allocated connections if all existing connections are being reused from a connection pool. To ensure that such Gateway daemons can be efficiently used in a failback situation, set the CICS TG `idletimeout` or the Java EE Connector architecture connection factory connection pool properties for aged timeout or unused timeout in WebSphere Application Server to ensure that connections are periodically recycled.

CICS TG WLM health

You can also increase the efficiency of your connection balancing function by enabling *health reporting* in the Gateway daemon. The Gateway daemon tracks the success rate of CICS requests and reports a health rating of 0 - 100 to WLM. The health rating can be used to influence the selection of a target Gateway daemon, when a new connection request is received, in combination with SEF.

Note: The health reporting function is influential when used in conjunction with the **SHAREPORTWLM** option, and is ignored when the **SHAREPORT** option is used.

You can enable the health reporting and set the frequency used to report the Gateway daemon health level to WLM using the `GATEWAY` section of the CICS TG configuration file, as shown in Example 7-1. The default health interval is 60 seconds.

Example 7-1 Configuring the Gateway daemon to enable health reporting to WLM

```
healthreporting=on
healthinterval=60
```

When CICS TG for z/OS is configured to report server-specific health values to WLM, the health values are dynamically calculated for a given Gateway daemon as a percentage value, based on the number of failing ECI and ESI requests in a configured interval. Over the intervals, the Gateway daemon monitors the following error codes to determine the health of communications with CICS:

- `ECI_ERR_NO_CICS`
- `ECI_ERR_RESOURCE_SHORTAGE`
- `ECI_ERR_SYSTEM_ERROR`
- `ESI_ERR_NO_CICS`
- `ESI_ERR_RESOURCE_SHORTAGE`
- `ESI_ERR_SYSTEM_ERROR`

The health values for each Gateway daemon will then be used by IBM Communications Server to prioritize the distribution of new socket connections by port sharing or Sysplex Distributor to Gateway daemons in the load balancing group. Gateway daemons reporting a higher health value receive a greater proportion of the incoming connections than those reporting a lower health value.

> **Note:** Health reporting is effective only in TCP/IP load balancing topologies with CICS TG running in remote mode, in combination with a WLM configuration.

Each Gateway daemon instance within a Gateway group must have health reporting enabled to achieve workload distribution based on the health metric.

Server-specific health can be used in conjunction with Dynamic Server Selection (described later in this chapter) to provide a mechanism to prevent *storm drain* scenarios by the removal of a specific Gateway daemon from IP connection balancing within the sysplex when all of its available CICS servers are offline (see Figure 7-6). In addition, because WLM also provides feedback on displaceable CPU capacity, it can be useful in allocating connections to LPARs where there is a sufficient general-purpose or a System z Application Assist Processor (zAAP) to run the required workload.

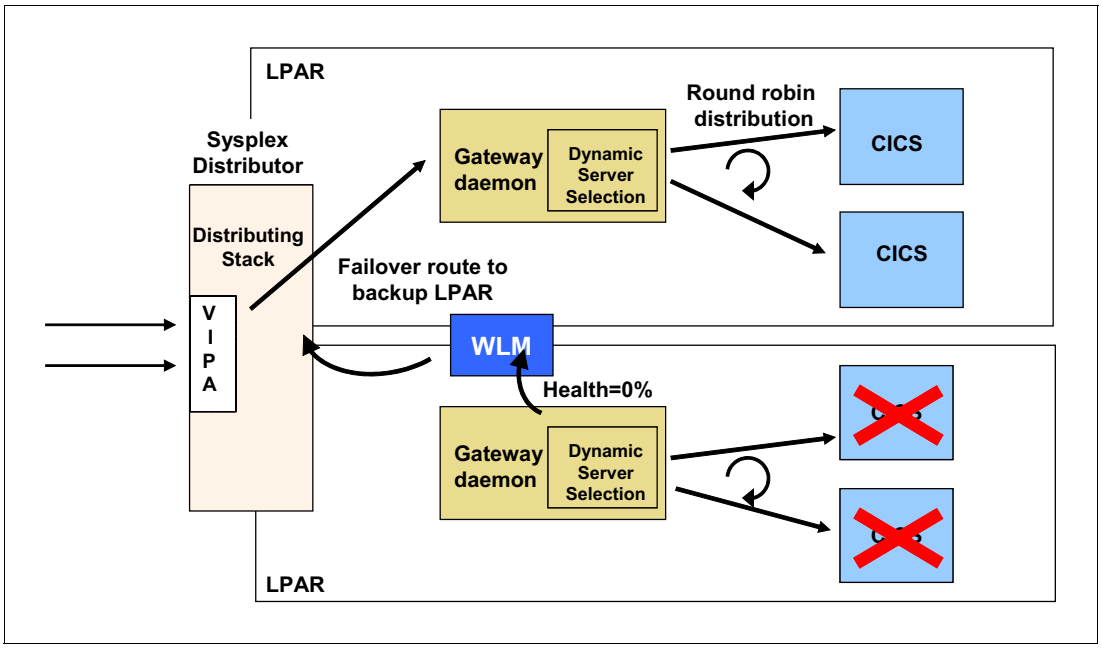


Figure 7-6 CICS TG and health reporting with port sharing

Be aware of the following issues when you use **SHAREPORTWLM** with CICS TG health reporting:

- The *Health value* reported by the CICS TG Gateway daemon statistic GD_CHEALTH is updated on an interval basis. The Health value reported by WLM is updated to match this value; however, this is only effective when the next ECI connection request is received on the shared port after the health interval has occurred.
- The normalized values of the WLM server-specific weights are displayed in the WLM "NormalizedWeight" value of NETSTAT reports. The original raw weights received from WLM are proportionally reduced for use by the distribution algorithm and they incorporate the TSR value. If **SHAREPORTWLM** is specified on the PORT profile statement, connections are distributed in a weighted round-robin fashion using these normalized weights.

- As the health of one or more Gateway daemons reduces, the WLM NormalizedWeight values are adjusted to be in proportion to the Health values. The NormalizedWeight value is used to determine how to proportionally distribute the requests received on the shared port. Therefore, the worse the health of one Gateway daemon, the fewer connections will be routed to it. When the NormalizedWeight value is zero, no more connections are routed to that Gateway daemon.

  Note: If the health of a Gateway daemon is significantly worse than other Gateway daemons listening on the same shared port, it is possible that the NormalizedWeight value might fall to zero while the Health value is nonzero (although, in this case, Health will normally be a low value, for example, 5).

- Because WLM will not route requests to a Gateway daemon whose NormalizedWeight is zero, manual intervention is required to recover the Gateway daemon:
  - Ensure that at least one of the CICS servers defined to the Gateway daemon is active and able to process requests successfully and that any resource limits encountered have been resolved.
  - Reset the health of the Gateway daemon using this command:

    ```
    F jobname,APPL=HEALTH,RESET
    ```

The next ECI request to the shared port will cause WLM to update its Health value for that Gateway daemon to 100 (to match the statistic value) and update its NormalizedWeight value to include this Gateway daemon back into the WLM distribution algorithm.

7.2.3 Multiplatform IP load balancing solutions

To avoid one of the SPOFs that a multiplatform system can have, you can benefit from various solutions that can be implemented either by the hardware, hypervisors, or operating system facilities.

A similar solution to z/OS TCP/IP shared ports can be implemented using external hardware TCP/IP balancers. This technology allows multiple Gateway daemons on independent operating system instances to receive connections through a common advertised address. If one Gateway daemon becomes unavailable, other instances can process the workload transparently to the user.

CICS TG supports the use of virtualization environments as defined by the Virtualization Policy for IBM Software. For more details, see the following technote:

http://www.ibm.com/support/docview.wss?uid=swg21304599

AIX workload partitions (WPARs) are one example of using virtualization to run multiple Gateway daemons on multiplatforms. In the next topic, we explore these AIX facilities in detail.

IBM workload partitions for AIX

The workload partitions (WPARs) provide a way to partition work within an AIX logical partition (LPAR). WPARs allow a single instance of AIX to be divided into multiple virtual partitions, so that the system administrator can quickly deploy multiple, isolated AIX environments without the overhead of managing individual AIX images. There are two types of WPARs:

- Application WPAR: A single process and its children run in their own partition.
- System WPAR: An entire AIX environment that can be either shared or detached.

If a system WPAR is shared, it mounts the global environment's `/usr` and `/opt` partitions as read-only and shares them. If a system WPAR is detached, it has a clone of the global environment, and all file systems are local to the WPAR. Figure 7-7 on page 113 shows a shared and detached system WPAR created in the same LPAR. The shared WPAR has read-only access to `/usr` and `/opt` in the global environment. The detached WPAR has no access to the global environment and maintains its own file systems.

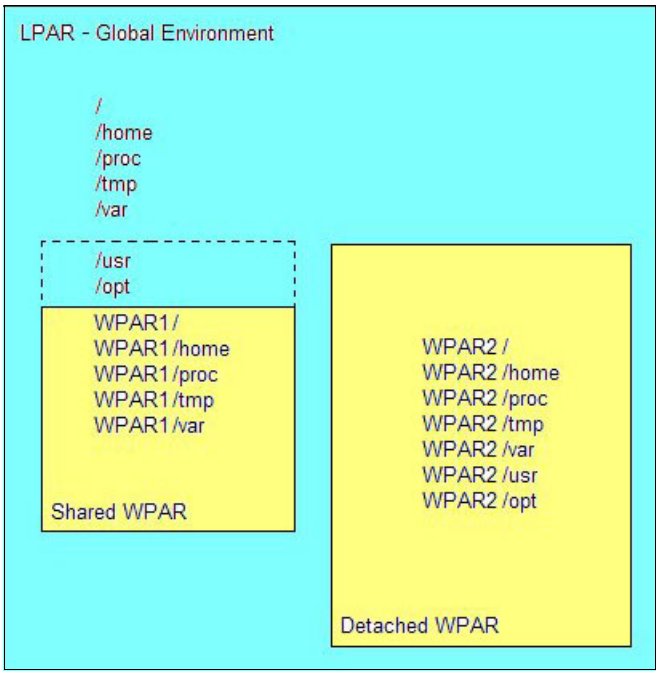


Figure 7-7 Deploying system WPARs

The following section explores the different types of WPARs and the ways in which you can use them when deploying IBM CICS TG on AIX.

Application WPARs

The simplest kind of WPAR is the application WPAR, which is a lightweight environment suitable for executing one or more application processes. CICS TG supports application WPARs for client applications running either in remote mode, or where the IPIC protocol is used, in local mode. Figure 7-8 on page 114 shows a Gateway daemon running in the global environment with three application WPARs, each running a different type of application connecting to that Gateway daemon.

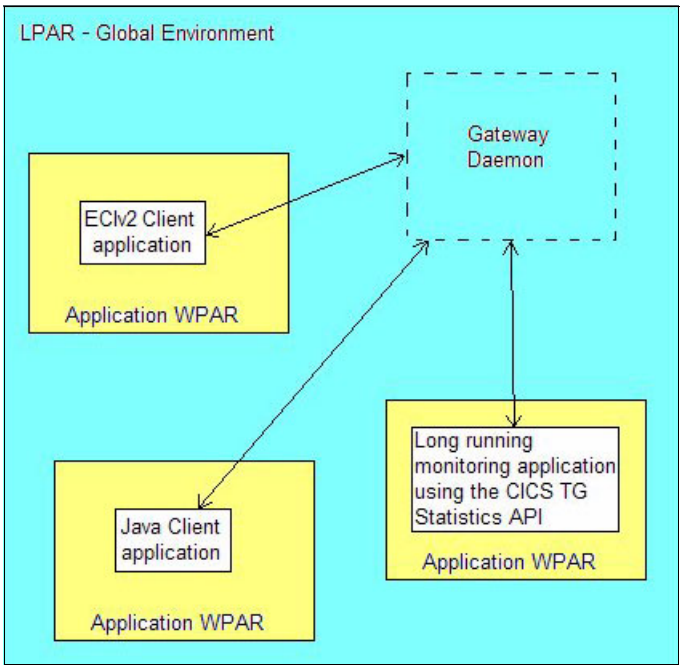


Figure 7-8 Deploying application WPARs

Application WPARs are appropriate when the client applications in application WPARs run in their own isolated environment, complete with their own network stack, and the application instances are easy to identify during monitoring. CICS TG can benefit, because a problem within an application does not affect the rest of the LPAR or any other WPARs running within it.

If you need to run multiple applications in the same LPAR, where multiple, long-duration applications are running in the same LPAR, you can run each application in its own WPAR. Doing so isolates each application from the other parts of the system, so that if an application fails, the failure is contained within the WPAR without affecting the other applications.

System WPARs (shared)

A shared system WPAR appears as an AIX instance, but it shares the global environment's `/usr` and `/opt` file systems and mounts them in the WPAR as read-only. An example of how to use system WPARs to run CICS TG is illustrated in Figure 7-9 on page 115. This example shows a CICS TG installation configured to run in the global environment with three shared system WPARs created within the global environment LPAR. The first system WPAR uses the `ctg.ini` configuration file in the global environment, and the other two WPARs have their own local configurations but still use the installed CICS TG executables.

If you need to expand the capacity of an existing LPAR that uses TCP/IP connections while maintaining the low administrative overhead of a single product installation and single configuration, you can benefit from the shared system WPARs.

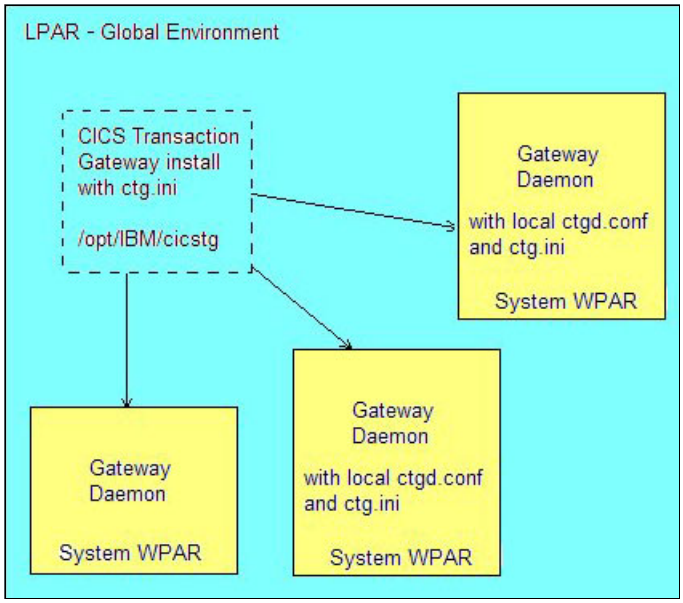


Figure 7-9 Deploying shared system WPARs

By default, in a shared system WPAR installation, the configuration files are shared across all the shared WPARs, because the configuration files are in the `/opt/IBM/cicstg/bin` directory. Sharing a configuration works well for TCP/IP connections to CICS, but for System Network Architecture (SNA) and Internet Protocol interconnectivity (IPIC), which require unique values to identify the gateway instance, sharing is restricted and only the first gateway using the configuration that connects to CICS can establish a connection. Other connections are rejected.

When you plan to run CICS TG as a background process using the **`ctgd`** process, this sharing limitation can be overcome by setting the **`CTGDCONF`** environment variable in each of the system WPARs to point to the local `ctgd.conf` that sets the **`CICSCLI`** environment variable to be the `ctg.ini` file in the WPAR, which contains unique identifiers for that instance of the CICS TG. The **`ctgd`** process checks for the presence of this environment variable before using the defaults. If **`ctgd`** is not used, and if **`ctgstart`** or **`cicscli`** is used, you must set the **`CICSCLI`** environment variable directly to point to the local `ctg.ini` file.

The main advantage of a shared system WPAR topology is that the system software, including CICS TG, the SNA Remote API Client, and the C/C++ compilers, are installed once in the global environment and are then available to all shared system WPARs without any additional installation. Shared system WPARs can therefore have a relatively small footprint when compared with WPARs that are not shared.

In shared system WPAR scenarios (for example, when communicating with CICS over TCP/IP), the configuration can also be common across all CICS TG instances. If Gateway daemons must be uniquely identified (for example, when using IPIC or SNA protocols to communicate with CICS), each WPAR must have its own local configuration, which must be managed independently of the global environment. Shared system WPARs also enable more efficient utilization of the LPAR and let you create a workload manager in the global environment to distribute work between the WPARs.

Detached system WPAR

A detached system WPAR is an entirely self-contained AIX installation, complete with its own isolated file system and network stack. An example of how you can use a detached system WPAR to run CICS TG is illustrated in Figure 7-10 on page 116.

Figure 7-10 shows CICS TG installed in the global environment, and three detached system WPARs. Unlike the shared system WPARs in the previous section, these installations are isolated, and the program and configuration files are installed separately.

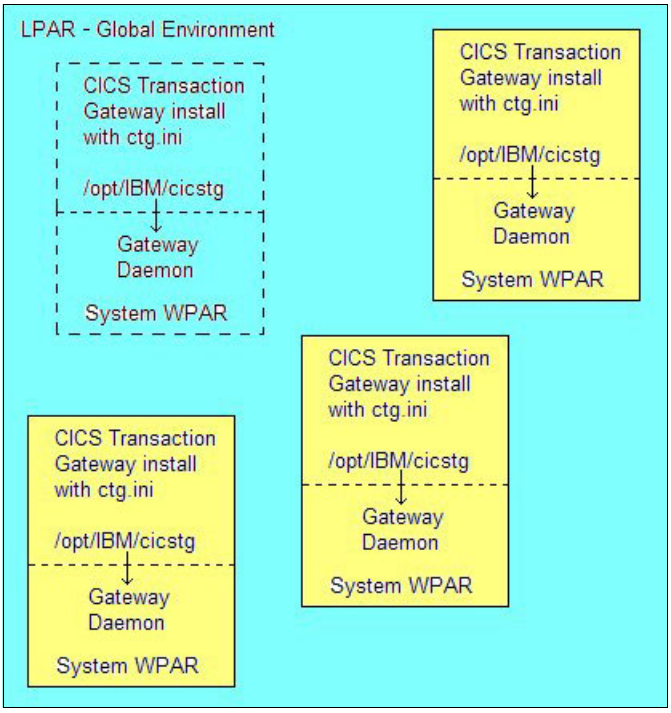


Figure 7-10 Deploying detached system WPARs

Detached system WPARs offer the benefits of LPARs but have a smaller processor and memory footprint, enabling you to further carve up LPARs to submanage resources while running multiple gateways in the same LPAR. This improved utilization of the LPAR enables you to create a workload manager in the global environment to distribute work between the WPARs.

If you need to run multiple versions of the CICS TG concurrently, with multiple machines or LPARs that are each hosting a different release of CICS TG, this topology is likely to use a large amount of machine resources or LPAR resources. A more efficient solution is to consolidate the hardware into a single LPAR with each instance running as a detached system WPAR.

If you also want to test a new release before you deploy it, you can use the detached system WPAR to test a new release of CICS TG in the current operating environment with existing applications. There is no impact on the existing installations and configurations. For configuration guidelines, see the article:

http://ibm.co/1kEaHAo

7.3 Dynamic server selection

This section highlights the SPOF exposure from the Gateway point of view, considering only the connection between the Gateway daemon and CICS TS. This section also includes the CICS TG answers to the exposures.

If the Gateway is only connected to a CICS region, and it is not available, the client is not able to process the request. Figure 7-11 shows how CICS TS can be a SPOF even if more than one Gateway daemon is active.

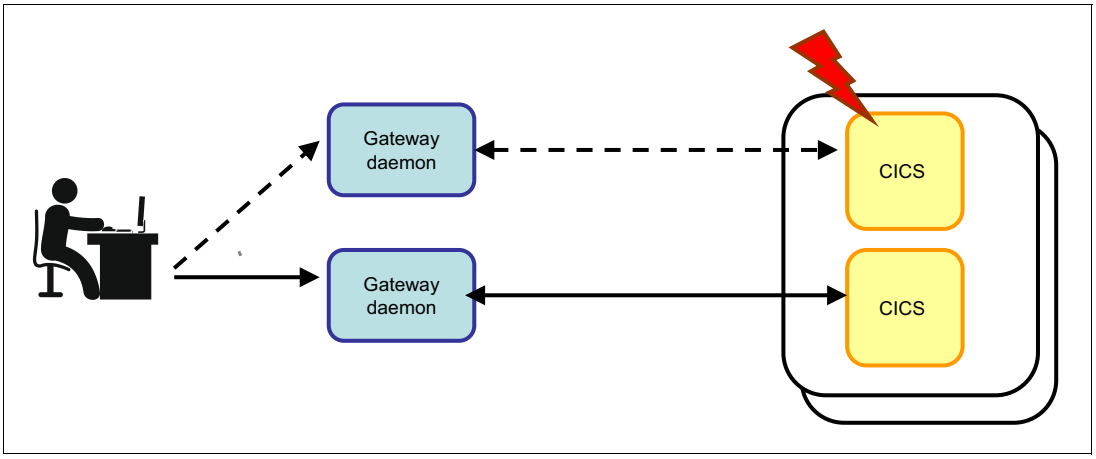


Figure 7-11 CICS as SPOF

The Gateway daemon is able to dynamically retry failed requests by retrying with an alternative CICS server, so the client application is not affected.

Dynamic server selection (DSS) consists of several CICS TG functions, providing the ability for the CICS TG run time to dynamically select a CICS server for any given ECI request. This feature allows each Gateway daemon to route requests appropriately, selecting a local CICS region or routing specific requests to a specialized group of CICS regions.

DSS allows the Gateway daemon to select the CICS server dynamically at run time for requests from client applications that do not need to know which server will process their requests, but which can be selected at run time.

DSS can be controlled dynamically using the CICS request exit in all the CICS TG products or can be implemented with other features described later.

7.3.1 Default server

If there is no server name in the ECI requests, you can assign a default server name at run time, using the **`defaultserver`** parameter in the `PRODUCT` section of the Gateway configuration file.

Figure 7-12 on page 118 shows an example of default server implementation.

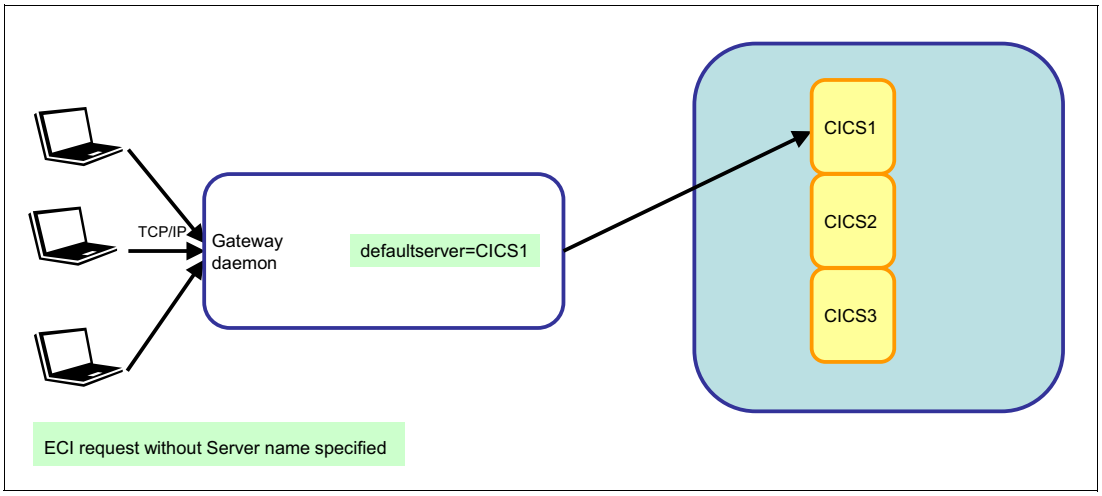


Figure 7-12 Example of default server implementation

7.3.2 The CICS request exit

The Gateway daemon provides an exit point that can call the CICS request exit to select a CICS server name for an ECI or ESI request. The CICS request exit can be used for request retry, DSS, and for rejecting invalid requests. If the server name returned by a CICS request exit is null, the request is sent to the default CICS server if one is specified in the configuration file.

The CICS request exit provides maximum flexibility for complex routing decisions, allowing custom-written server selection logic.

A fully functional working sample that uses text-based configuration files is provided in CICS SupportPac CA1T. This SupportPac provides the capability to configure workload management via round-robin or failover polices. It offers primary and secondary levels of failover and configurable timeouts to enable failback to primary CICS regions after they are restarted and online policy updates. It is also possible to request validation rules.

To download the SupportPac, see this website:

http://www.ibm.com/support/docview.wss?uid=swg24021484

Figure 7-13 on page 119 shows a scenario of the CICS request exit implementation with round-robin and failover implementation. The platform where the Gateway daemon is installed is not important.

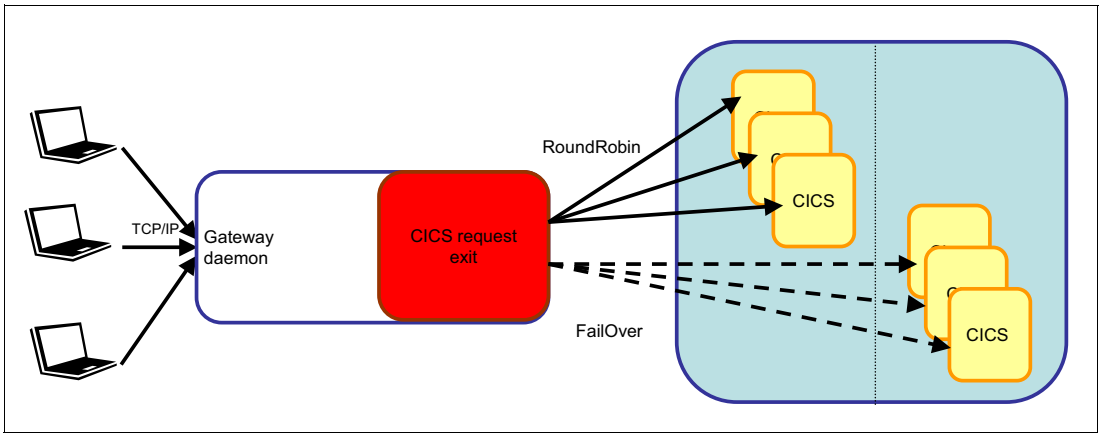


Figure 7-13 Shows an example of a CICS request exit implementation

Note: Transaction ID, program name, and user ID are also available to the CICS request exit for XA transactions, as of Version 9.0.

7.3.3 DSSPOLICY (z/OS only)

DSSPOLICY is a feature of the CICS TG for z/OS product. It is policy-based DSS that supports generic and server-specific rules, and a choice of round-robin or failover algorithms.

DSSPOLICY is an alternative to CICS request exit for which no programming skills are required. It is quick and easy to implement in the Gateway daemon configuration file, using DSSGROUP and DSSPOLICY definitions.

Note: Java programming skills are not required to use the sample CICS request exit program provided in SupportPac CA1T.

You can map your CICS servers to groups and apply a selection algorithm to them. The selection algorithm can be round-robin or failover, depending on your goal. The DSSPolicy maps application requests to a DSSGROUP. This implementation allows you agile and easy management of your routing selection.

Note: `LOGICALSERVER` section definitions for CICS TG for z/OS are deprecated and superseded by policy-based DSS definitions.

Figure 7-14 on page 120 shows a sample implementation of DSS. In this sample, there are two groups: Group1 routes all the requests for CICSX server and Group2 routes all the requests without a specified server name. Group1 uses the round-robin algorithm and Group2 uses the failover algorithm. The example shows ECI requests arriving with no specified server. All requests will be sent to CICSA3 or CICSA4 if CICSA3 is not available.

The ECI request arriving with server name CICSA4 will not be affected. The ECI request arriving with server name CICSX will be round-robin routed to CICSA1, CICSA2, and CICSA5.

This example shows the behavior of an ECI request, but the same behavior can be applied also in Extended ECI and XA requests. DSS will retry failed requests around all defined servers in this list, therefore hiding CICS failures from the client system.

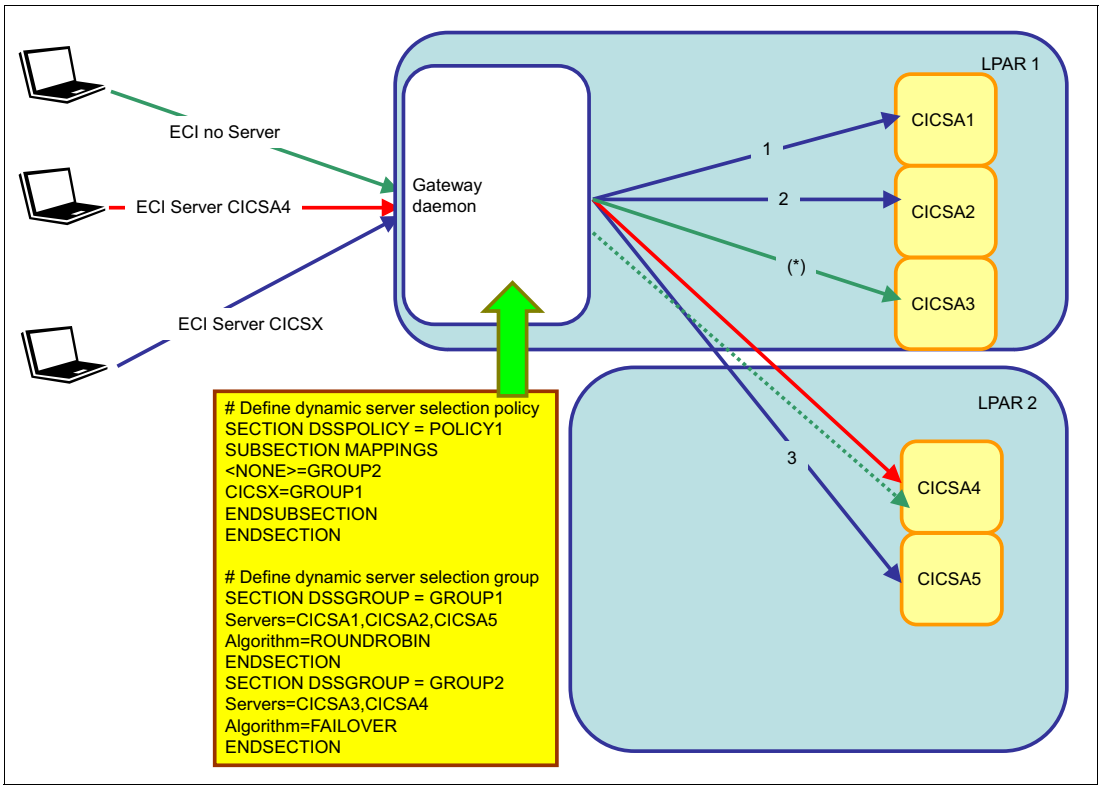


Figure 7-14 Sample DSSPolicy scenario

DSSPolicy information has the following limitations compared to the CICS request exit (CA1T):

- No ability to refresh configuration while the Gateway daemon is running
- No ability to switch policy while the Gateway daemon is running

7.3.4 DFHXCURM (z/OS only)

The CICS EXCI user replaceable module, DFHXCURM, also provides the ability to retry failed ECI requests. However, its usage is no longer suggested with CICS TG, because it does not support IPIC connections and is not integrated with CICS TG statistics and systems monitoring.

7.3.5 Client API user exits

The Client daemon provides user exits to modify the target CICS server for ECI and EPI requests destined to CICS servers using the TCPIP or SNA protocol drivers.

Note: At least for remote mode ECI applications using the Gateway daemon, it is advised to use the CICS request exit instead of Client API exits, because the CICS request exit is better integrated with the CICS TG statistics and request monitoring facilities.

For further details, see the *ECI Client API exits* and the *EPI Client API exits* topics in the CICS TG for Multiplatforms Information Center:

http://ibm.co/1ldvUkI

7.3.6 Client daemon Workload Manager on Windows

The Windows versions of the CICS TG for Multiplatforms and CICS TG Desktop Edition products include a feature called Workload Manager (WLM). It is possible to use Workload Manager to distribute work across multiple CICS servers, but only if the CICS servers are connected with the TCPIP or SNA protocols.

Note: At least for remote mode ECI applications using the Gateway daemon, it is suggested to use the CICS request exit instead of WLM, because the CICS request exit is better integrated with the CICS TG statistics and request monitoring facilities.

WLM allows you to distribute work across available and equivalent CICS servers, and also to favor one CICS server over another (*biasing*). It is also possible to handle requests that fail to reach a target CICS server and to redirect them.

WLM implements a classification of the CICS server in server groups (it is also possible to insert the program names that have to run in a specific server group).

WLM has two algorithms to route the ECI requests:

- The round-robin algorithm assumes that all server groups are equally valid for selection.
- The biasing algorithm provides a way of balancing workload by specifying that workload distribution. A server group can be a favorite and it will receive more work. For example, if there are two server groups with a bias of 75 and 25, program requests are sent in a ratio of 3:1 to the first server group.

If a request to route an ECI call to a particular CICS server fails, indicating that the server is no longer available, the server is removed from the list of available servers.

If a request fails to reach the target CICS server, the WLM tries again, this time sending it to a different server that it knows is available. The server that failed is removed from the list of active servers. It is added back after a period of time, which you specify in the server group timeout field. If the server fails again, the process is repeated.

WLM is able to maintain the affinity created by an extended Logical Unit of Work (LUW), routing all the requests to the same CICS server.

Note: WLM cannot be used with servers that are connected using the IPIC protocol.

Figure 7-15 on page 122 shows a scenario with a Windows WLM implementation containing two server groups using a round-robin distribution of the workload.

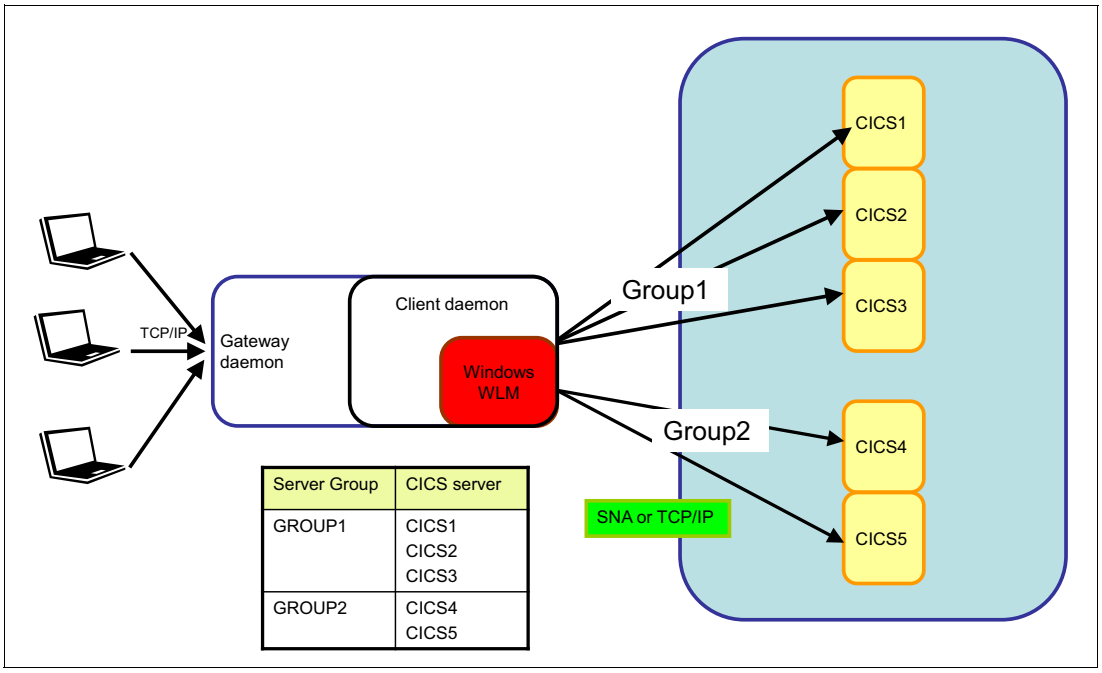


Figure 7-15 Shows an example of Windows WLM implementation

7.4 Importance of scaling and growing in a continuous availability system

A highly available CICS TG configuration enables you to increase throughput in line with an increasing workload. The HA implementation allows you to scale your environment easily, because the infrastructure is already in place.

The workload and the amount of data transmitted for each payload can trigger the need to scale.

Another important consideration is the capacity on which the CICS TG must normally run. Theoretically, in an HA implementation, a single region is able to process all the incoming requests as a failover backup.

You need to evaluate the maximum risk exposure that you will accept, compared with the MIPS available, and tune the workload capacity of each Gateway.

CICS TG for z/OS provides a highly optimized inbound CICS connector infrastructure for a wide variety of clients. The key factor affecting the scalability of a CICS TG solution is usually the payload size of each request. The amount of data transmitted for each payload depends on both the application design and whether COMMAREA or channel-based requests are sent to CICS:

- When using channel-based requests, only containers within the channel that have been modified are returned back to the calling application. Therefore, an efficient design ensures that different containers are used for input and output to ensure that only the data required is transmitted.
- When using COMMAREA-based requests, CICS provides the ability to dynamically truncate the inbound or outbound data flows that contain trailing blank (null) data.

A two-phase commit provides an optimized mechanism for supporting distributed units of work between Java EE application servers and CICS, with a small delta cost per transaction. However, as with any use of distributed transactions, the number of interactions during the scope of the unit-of-work (UOW) can have significant effects on scalability, because CICS transactions will remain active longer and locks will be held longer on recoverable resources.

Using Secure Sockets Layer (SSL) for security can add overhead both to the process of connection establishment and also to the encryption of each request. The first priority is to ensure that connections are reused wherever possible, because the most significant overhead is the SSL handshake during connection establishment. Additionally, usage of SSL cipher suites, such as Data Encryption Standard (DES), Advanced Encryption Standard (AES), or Triple DES, that are supported using the System zCentral Processor Assist for Cryptographic Function (CPACF) hardware, ensures that the cost of encryption is as low as possible.

Limiting factors on z/OS

CICS TG provides a wide range of facilities to help applications scale when faced with increasing system load. However, there are key considerations that need to be taken into account to ensure Gateway daemons can scale to their maximum extent:

- Multitasking

  A single CICS TG region has the potential to run hundreds of threads (TCBs) that can run in parallel with other applications to fully use available processors. So, there is a physical limit in scalability by the capacity of a single CPU and the frequency at which other applications give up control.

- Optimizers and appliances

  CICS TG provides the ability to use a variety of optimizers and appliances to improve scalability, including the following range of options:

  – System z Application Assist Processor (zAAP)

    zAAP processors are speciality processors to offload Java workloads from the System z general-purpose central processors (CPs), with the aim of reducing processing costs. A significant amount of CICS TG code is written in Java, and IPIC connections are the largest users.

  – System z Integrated Information Processors (zIIPs)

    zIIP processors are similar in concept to zAAPs but they are designed to offload TCP/IP and XML processing.

    > **Tip:** z/OS V1.11 added a new capability that enables zAAP-eligible workloads to run on the zIIP. This new capability is ideal for clients without enough zAAP-eligible workload or zIIP-eligible workload to justify a specific speciality engine.

  – Cryptographic coprocessors, such as the CPACF and CEX2, can be used to accelerate public and private key encryption.

You can also benefit from the facilities provided by CICS Configuration Manager to scale your CICS TG implementation easily because CICS Configuration Manager is able to discover and clone CICS TG regions.

7.5 Considerations for XA transaction support on z/OS

This section introduces basic concepts related to XA transactions. An *XA transaction* is a recoverable unit of work performed by one or more resource managers in a distributed transaction processing environment, which is coordinated by an external transaction manager where the Extended Architecture (XA) protocol is used for syncpointing and recovery.

In CICS TG for z/OS, the use of XA transactions together with high availability features requires the creation of a highly available Gateway group, formed of Gateway daemons with matched capabilities, and access to Resource Recovery Services (RRS). In combination with highly available Gateway groups spanning a Parallel Sysplex, XA transactions can be used transparently with CICS TG Dynamic Server Selection (DSS) features, enabling both workload distribution and failover capabilities. Figure 7-16 shows the potential scope of an XA transaction.

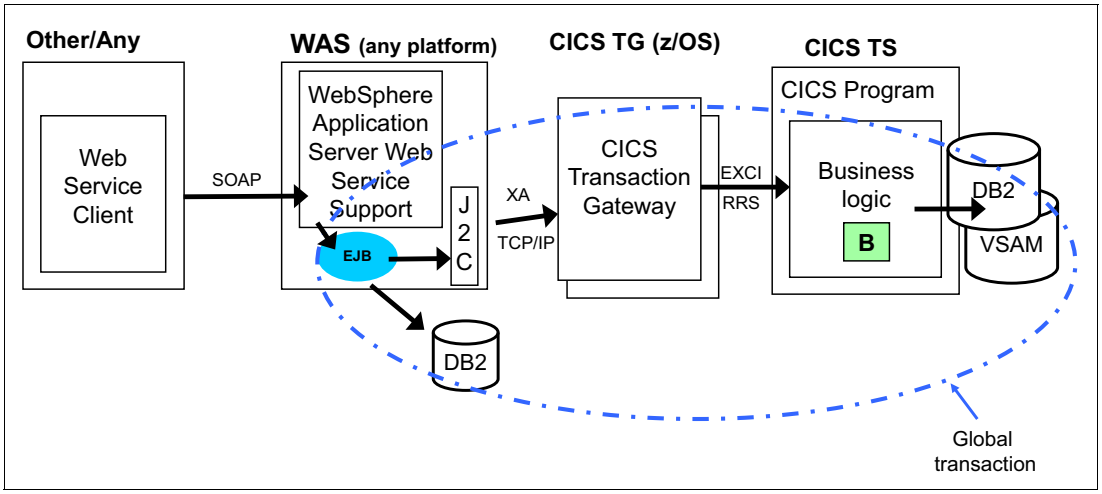


Figure 7-16 HA transaction example

In a high availability configuration, you must consider many aspects. For a full understanding, see the example situations in the next section.

Figure 7-17 on page 125 shows the normal flow of an XA transaction: a transaction starts, flows some ECI requests, and then commits, proceeded by a prepare commit. Each participant in the XA transaction needs to respond positively to the prepare commit to commit successfully.

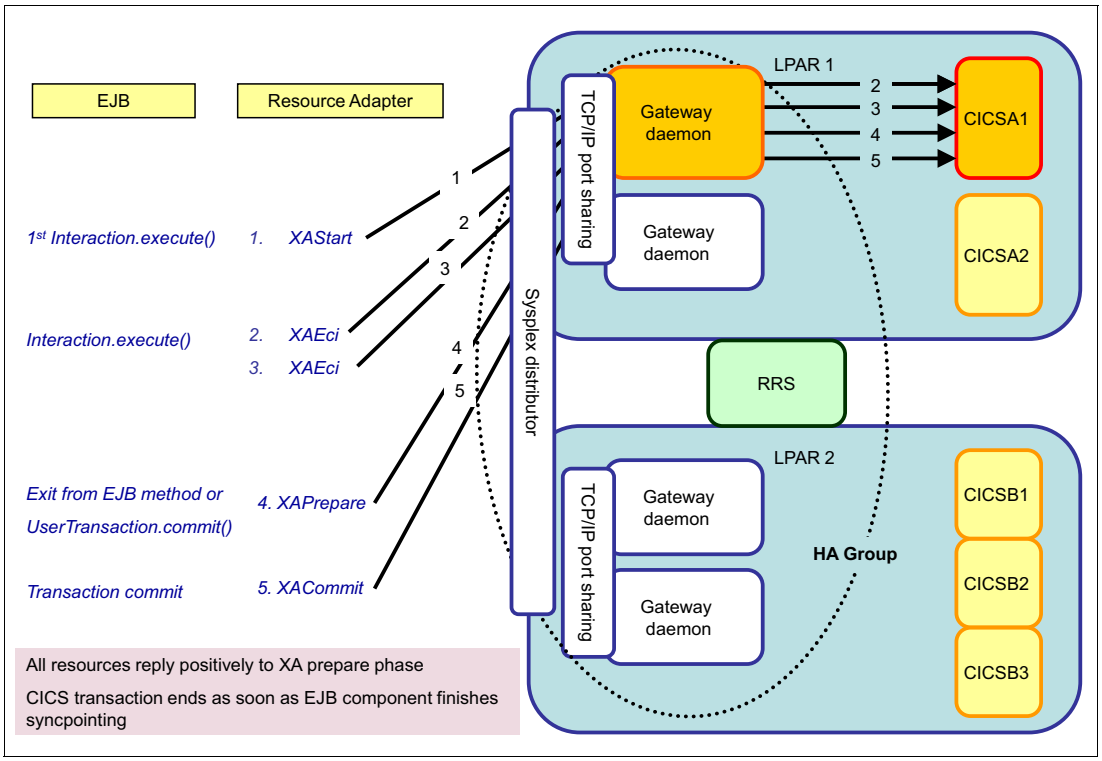


Figure 7-17 Normal case of an XA transaction

If something goes wrong, and you need to roll back an in-flight XA transaction, the transaction coordinator needs to guarantee the data integrity. All the resources must be rolled back, but if the XA rollback cannot flow (connection lost), it will not cause a problem because CICS will roll back when the transaction ends abnormally.

Figure 7-18 on page 126 shows the logical flow of an in-flight failure. The failure is communicated to the transaction manager, which will evaluate the actions required to guarantee the data integrity.

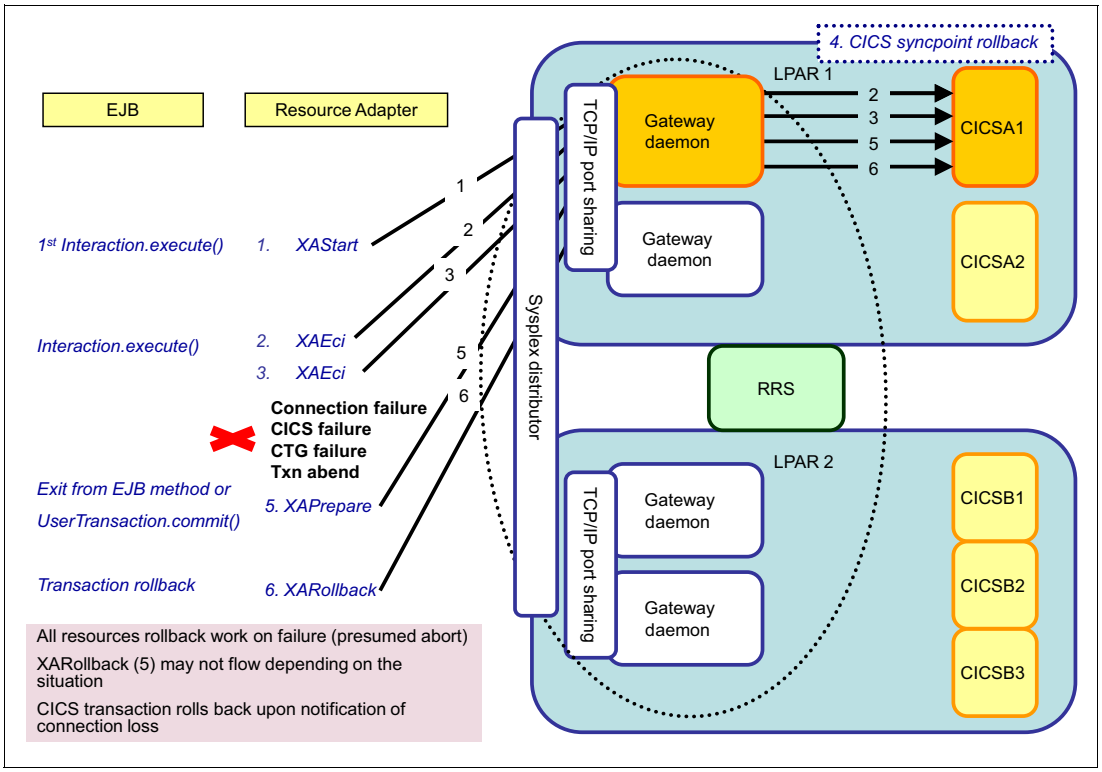


Figure 7-18 In-flight failure of XA transaction

When a failure occurs in the middle of a sync point processing (for example, between prepareCommit and XAcommit), an in-doubt UOW situation arises because the coordinator cannot know the actual status of the unit of work. The resolution of the in-doubt situation will happen when the connection is reestablished.

> **Note:** CICS can independently select the action to be taken when a task is in *in-doubt* status. The parameter **WAIT**=YES in the transaction definition specifies whether a unit of work is to wait, pending recovery from a failure that occurred after it had entered the in-doubt period, before taking the action specified by ACTION.
>
> Consider the heuristic decision carefully, evaluating also the actions of other transaction managers.

A Gateway group that uses TCP/IP load balancing can be viewed as a single logical Gateway daemon. A Gateway daemon instance in an HA group can recover in-doubt XA transactions on behalf of another Gateway daemon within the HA group. Figure 7-19 on page 127 shows an example of in-doubt processing.

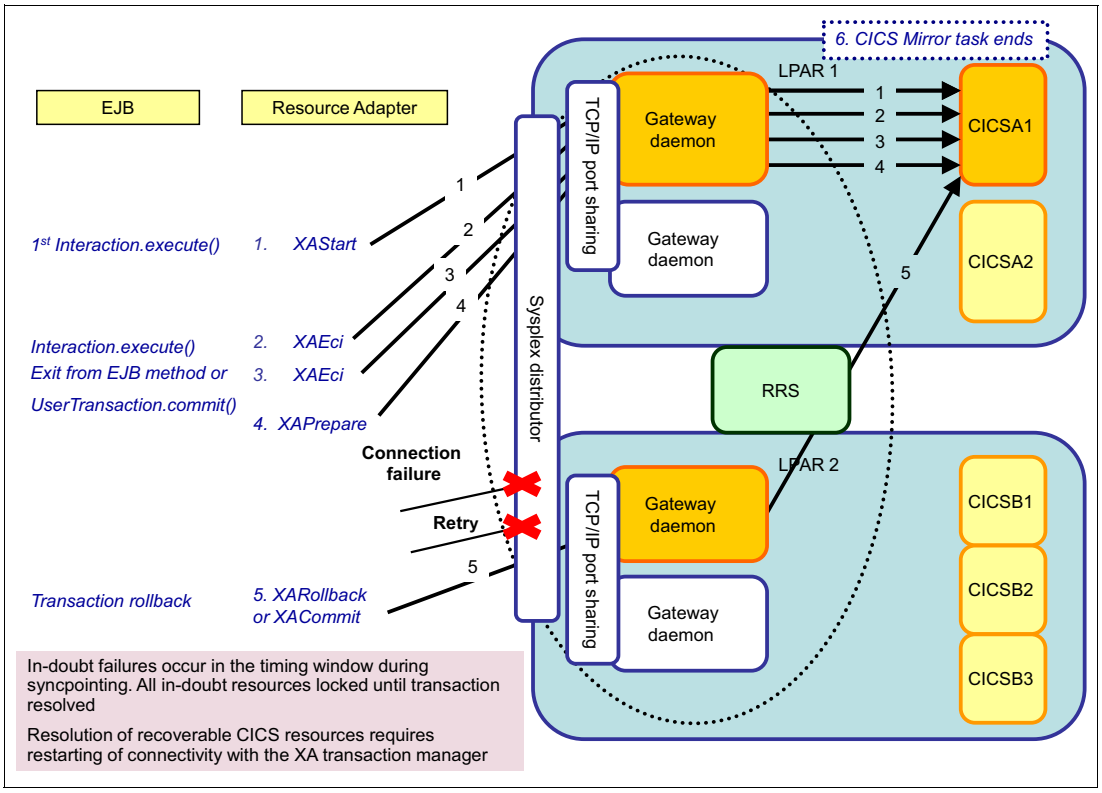


Figure 7-19 In-doubt situation

DSS is also integrated with the two-phase commit XA transaction support provided by CICS TG. DSS is designed to ensure that, after a CICS server has been dynamically selected for any given ECI request, all future requests within the scope of the UOW will be routed to the same CICS server until the UOW completes. Figure 7-20 on page 128 shows an example of in-doubt processing.

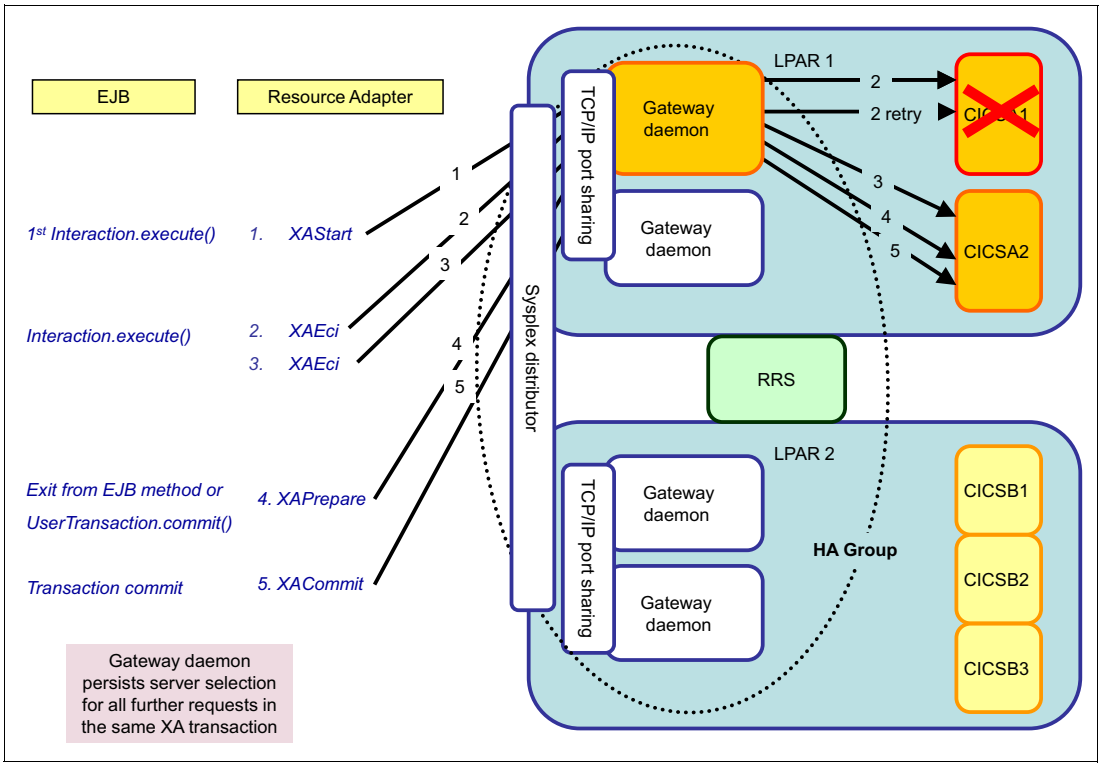


Figure 7-20 Dynamic server selection and XA transactions

There are certain situations that leave some residual data in RRS and a Gateway cold start might be needed. If one Gateway issues the XAPrepare and the connection between the Gateway daemon and the CICS server is lost, another Gateway can take part in the XA transaction, issuing the **`XARollback`** or **`XACommit`** to the CICS that is enlisted in the XA transaction. Figure 7-21 on page 129 shows an example.

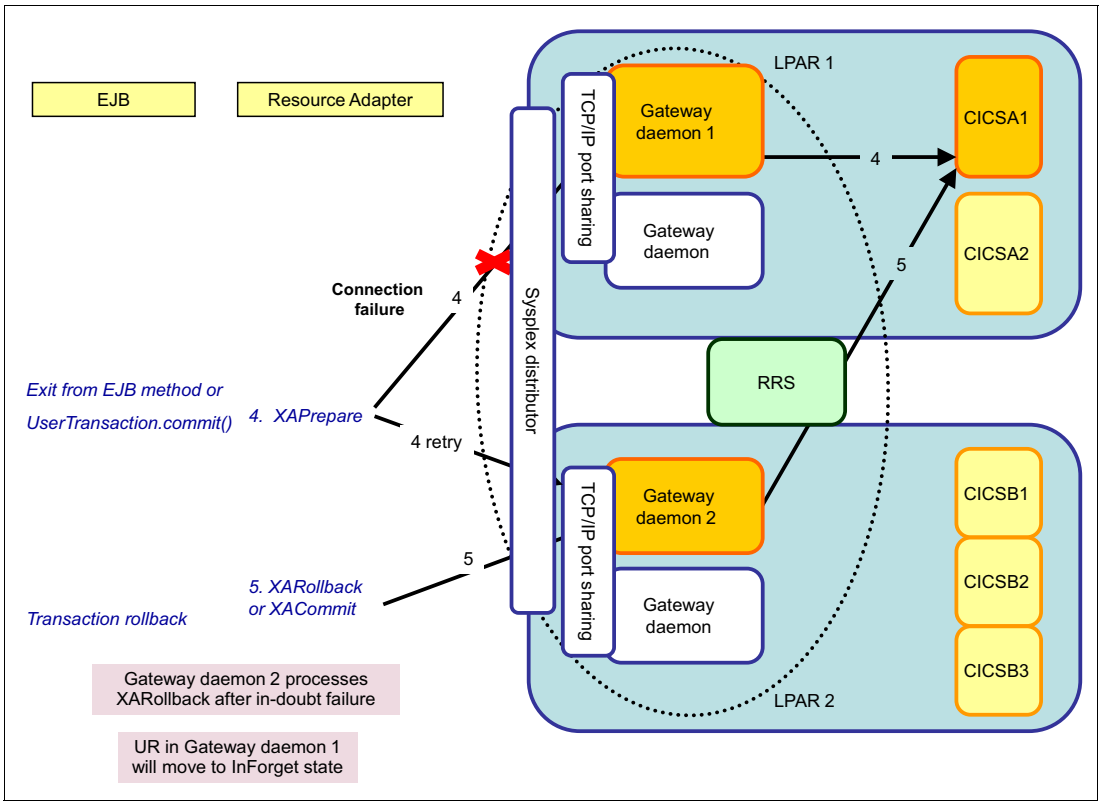


Figure 7-21 Situation needs a cold start

The implication of CICSPlex SM routing

If IBM CICSPlex® SM is in use for routing distributed program link (DPL) requests from routing regions to application-owning regions (AORs), CICS UOW affinity support is used to ensure that all requests within an XA transaction are routed to the same AOR to prevent potential transactional deadlocks.

These transactional deadlocks can occur if two requests within the same distributed UOW are dynamically routed to two different AORs, but subsequently access the same recoverable resource, such as a shared Virtual Storage Access Method (VSAM) file. If this occurs, the second request in the UOW will suspend, because it will wait for a lock when trying to update the shared resource, which has already been locked by the first CICS task (see Figure 7-22 on page 130).

Note: To avoid transactional deadlocks, it is often necessary to ensure that all distributed program link (DPL) requests that are dynamically routed to remote regions are routed to the same AOR if they are within the same distributed UOW. CICS TS V4.2 provides built-in support for this through CICSPlex SM WLM support for UOW affinities.

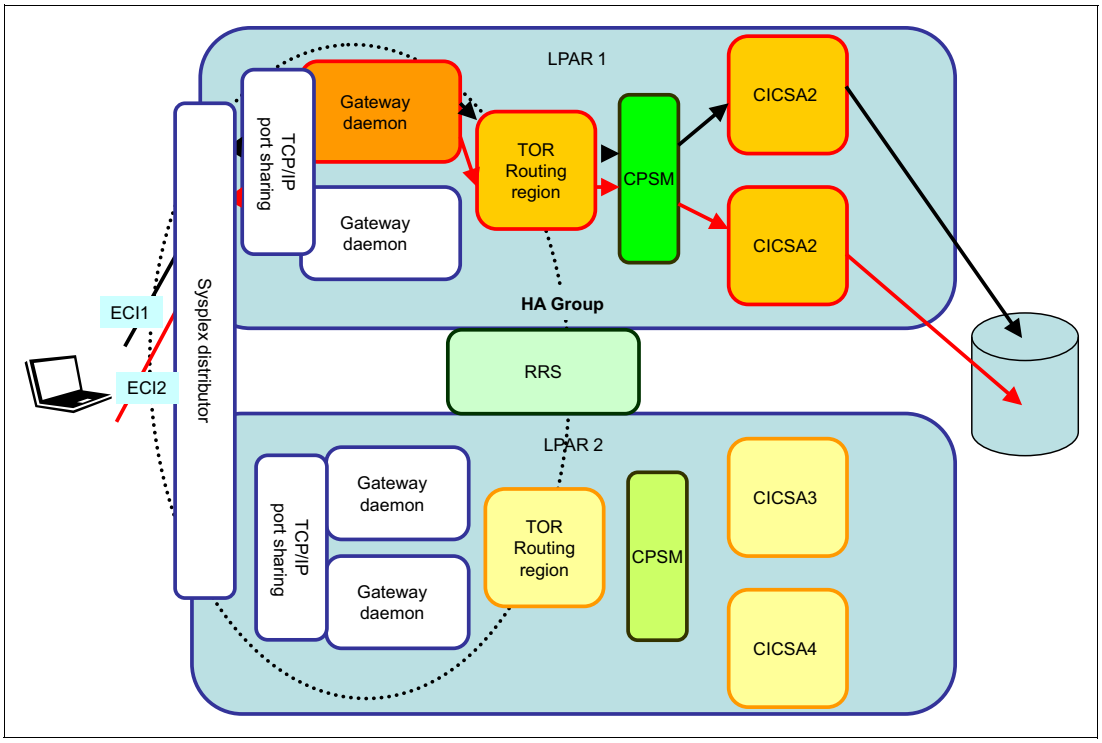


Figure 7-22 Possible deadlock situation

Impact of XA transactions on an HA solution

These are the details of the impact of XA transactions on an HA solution:

- If ECI requests are to be routed from a Gateway daemon to a CICS region on a different LPAR, IPIC connections must be used because the external CICS interface (EXCI) does not support the use of Resource Recovery Services (RRS) for transactional requests sent between LPARs.
- Specify one or more `IPICSERVER` section server names or application identifiers (APPLIDs) of EXCI-connected servers in a comma-separated string. If XA support is enabled, `IPICSERVER` section server names cannot be mixed with APPLIDs of EXCI-connected servers within the same DSS group.

8

Security

This chapter introduces the security considerations when connecting applications to CICS programs with CICS Transaction Gateway (CICS TG). It then describes how the CICS TG addresses the issues of authentication, authorization, data integrity, and confidentiality. It also examines how CICS and CICS TG work together with an external security manager, such as Resource Access Control Facility (RACF), to provide these facilities. Details are included that describe the mechanisms used to secure connections and to transmit security credentials to the CICS system.

This chapter covers the following topics:

8.1 Security basics

Effective identity management, authorization, and access control capabilities are the essential elements of a comprehensive enterprise security program. These security services are necessary to control the access to resources and sensitive information within the Information Technology (IT) environment.

8.1.1 The components of a security system

A complete security system puts the necessary infrastructure in place to provide a core set of capabilities:

- Identification
- Authentication
- Authorization
- Confidentiality
- Auditing
- Nonrepudiation

This section describes the purpose of each capability, and provides some context within a typical CICS system.

Identification

This is the ability to assign an identity to the entity accessing the system. Typically, the identity is used to control access to resources. Depending on the security model in which the identification is performed, the identity can be called a user ID, a UID, or a principal.

In general, a CICS user is an entity (a person or a program) that is identified by a user identifier (or user ID). All CICS users must be defined to the External Security Manager (ESM). When the security manager is RACF, information about each user is stored in a user profile.

Authentication

Authentication is the process of validating the identity claimed by the user. Authentication is performed by verifying the authentication information provided by the claimed identity. The authentication information is generally referred to as the accessor's *credentials*. A credential can be the accessor's name and password or it can be a token provided by a trusted party, for example, a Kerberos ticket or an X.509 certificate.

CICS supports several schemes for authenticating users. In many cases, the user's authenticity is verified by checking a password or password phrase supplied by the user. Depending on topology, authentication can be based on the user ID passed with the External Call Interface (ECI) request, a Secure Sockets Layer (SSL) client certificate, or a distributed identity (identity propagation).

Authorization

Authorization is the process of checking whether an identity, which has already been authenticated, must be provided access to a resource that it is requesting. A typical implementation of authorization is to pass a security context, which contains the identity that has been authenticated, to the access control mechanism.

In conjunction with the ESM, CICS provides a variety of security and control mechanisms. These mechanisms can limit the activities of CICS users to only those functions that any particular individual user is authorized to use. CICS provides three types of authorization checks to protect its applications, the resources they use, and the operational environment:

1. Transaction security ensures that users that attempt to run a transaction are entitled to do so.
2. Resource security ensures that users who use CICS resources are entitled to do so.
3. Command security ensures that users who use CICS system programming commands are entitled to do so.

To provide the necessary security for your CICS systems, CICS uses the Multiple Virtual Storage (MVS) System Authorization Facility (SAF) to route authorization requests to an ESM, such as RACF, at appropriate points within CICS transaction processing. The user's identity is passed on this call and the ESM return code indicates whether the user is authorized to perform that function.

Confidentiality

Confidentiality ensures that an unauthorized party does not obtain the meaning of the transferred or stored data. Typically, confidentiality is achieved by encrypting the data.

CICS and CICS TG support the SSL transport protocol to ensure that information remains private as it passes over the connection. The data exchanged between the sender and receiver is encrypted and only the client and the server can interpret the information.

Auditing

Auditing can be used to maintain a record of security-related events, such as a user signing in to a system or out of a system, or the details of who modified a given resource, and when. The provision of an audit trail is typically mandatory in enterprise organizations, allowing post-mortem analysis to be conducted in a security breach.

Auditing is a function of the External Security Manager on z/OS. A typical implementation of auditing is the RACF auditor tool. For more information about the RACF auditor tool, see *z/OS Security Server RACF Auditor's Guide*, SA22-7684:

http://ibm.co/1rxQ65V

Integrity

Integrity ensures that transmitted or stored information has not been altered in an unauthorized or accidental manner. Typically, it is a mechanism to verify whether what is received over a network is the same as what was sent.

Nonrepudiation

Nonrepudiation means that a data sender and a data receiver are able to provide legal proof to a third party that the sender did send the information and that the receiver received the identical information. Neither side is able to deny.

The IBM Redbooks publication, *Securing Access to CICS Within an SOA*, SG24-5756, contains a more thorough description of the security functions and security considerations when accessing CICS programs from client applications.

8.1.2 The role of cryptography

In computer security, cryptography provides the following processes:

- Encrypting converts usable data into a ciphertext, which conceals the meaning of the data from any unauthorized recipient.
- Decrypting converts ciphertext back to usable data.
- Hashing uses a one-way (irreversible) calculation to condense a long message into a compact bit string called a *message digest*.
- Generating a *digital signature* involves encrypting a message digest with a private key to create the electronic equivalent of a handwritten signature.

  A digital signature is used to verify the identity of the signer. This verification ensures that nothing has been altered in the signed document from the time that it was signed, and is a basis for nonrepudiation.

Most cryptographic systems combine two elements:

- An algorithm that specifies the mathematical steps that are required to encrypt the data.
- A cryptographic key (a string of numbers or characters) or keys.

The algorithm uses the key to select one relationship between plaintext and ciphertext out of the many possible relationships that the algorithm provides. The selected relationship determines the composition of the algorithm's result.

8.2 Network security

In a 3-tier system, remote mode applications communicate with a Gateway daemon over a TCP/IP network connection. The Gateway daemon might then use a further network connection to communicate with a CICS server. Both of these network connections can be secured with SSL for Java, JCA, and .NET applications, when using IPIC connections to the CICS server.

Implementing SSL with data encryption provides confidentiality of the messages transmitted on the connection. SSL also offers client authentication based on digital certificates.

8.2.1 Secure Sockets Layer and Transport Layer Security

CICS supports several security protocols to provide secure communication over a TCP/IP network:

- SSL 3.0
- Transport Layer Security (TLS) 1.0
- TLS 1.1
- TLS 1.2

Note: CICS Transaction Server for z/OS V5.1 (CICS TS) APAR PM97207 provides support for TLS 1.1 and TLS 1.2.

In this book, the term *SSL* is used as a general reference to these protocols, unless explicitly qualified.

The SSL protocol operates between the application layer and the TCP/IP layer. This allows it to encrypt the data stream itself, which can then be transmitted securely, transparent to the application layer. SSL also provides a means for establishing the identity of one or both participants in the connection.

Authentication occurs at connection time, and it is independent of the application or the application protocol. With SSL, authentication is performed by an exchange of certificates, which are blocks of data in a format described in the International Telecommunication Union (ITU) Telecommunication Standardization Sector (ITU-T) standard X.509. The X.509 certificates can be issued and digitally signed by an external authority known as a *certificate authority*.

For more information about the X.509 standard, see the ITU-T Recommendation on Public-key and attribute certificate frameworks:

http://bit.ly/1n2jC2K

8.2.2 Digital signatures, certificates, and key rings

SSL uses digital signatures and digital certificates for establishing a trusted relationship between a sender and a receiver of information sent over a network connection.

Digital signature

A *digital signature* is a unique, mathematically computed signature that demonstrates the authenticity of a transmission.

Digital certificate

A *digital certificate* allows unique identification. It is essentially an electronic ID card, issued by a trusted third party, which is known as a *certificate authority*. Digital certificates form part of the X.509 protocol. This framework provides for authentication across networks. A digital certificate serves two purposes: it establishes the owner's identity and it makes the owner's public key available.

A digital certificate contains the following information:

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

A digital certificate alone is not proof of an identity; it allows verification of the owner's identity, by providing the public key needed to check the owner's digital signature. Therefore, the digital certificate owner must protect the private key that belongs with the public key in the digital certificate. If you send your digital certificate containing your public key to someone else, your private key prevents that person from misusing your digital certificate and posing as you. If the private key is stolen, anyone can pose as the legitimate owner of the digital certificate.

Certificate authority (CA)

A digital certificate is issued by a CA and has an expiry date. When requesting a digital certificate, you supply your distinguished name. The digitally signed certificate includes your distinguished name and the distinguished name of the CA. This allows verification of the CA.

To communicate securely, the receiver must trust the CA that issued the certificate that the sender is using. Therefore, when a sender signs a message, the receiver must have the corresponding CA's signer certificate and public key designated as a trusted root key. Your web browser has a default list of signer certificates for trusted CAs. If you want to trust certificates from another CA, you must receive a certificate from that CA and designate it as a trusted root key.

Key ring

A *key ring* is a repository containing public keys, private keys, and digital certificates used by a network communications security protocol, such as SSL. Each certificate consists of at least a public key, and metadata related to the identity that it represents, such as a distinguished name.

SSL requires access to key rings for the establishment of secure connections. In the Java Secure Socket Extension (JSSE) implementation of SSL, the key ring is known as the *Java KeyStore (JKS) file.*

The SSL handshake

When an SSL connection is established, the client and server agree on which version of the SSL/TLS protocol they will use. They select a cipher suite, optionally authenticate each other, and use public key encryption techniques to generate a shared value to be used as the seed for encrypting and decrypting transmitted data.

The SSL handshake consists of several phases:

- Server authentication

  In the first phase, the server responds to a client's request by sending its certificate and cipher preferences.

  > **Note:** The encryption algorithm and key strength are selected by the client and server during the handshake. It is possible to negotiate to plaintext because two of the defined cipher suites specify no encryption.

- Client authentication (optional)

  The server requests that the client identify itself during the SSL handshake by providing its client certificate. Client authentication can only be requested by the server.

- Key negotiation

  The client then generates a master key, which it encrypts with the server's public key, and then transmits the encrypted master key to the server. The server authenticates itself to the client by returning a message authenticated with the keys derived from the master key.

Subsequent data is secured using the cipher suite that was negotiated during the handshake. Figure 8-1 on page 137 shows the SSL handshake without client authentication.

Client		Server
Client starts SSL handshake; sends list of supported ciphers	→	Server selects cipher
Client authenticates server's certificate and gets server's public key	←	Server sends its certificate to client
Client creates key exchange message and encrypts with server's public key; sends to server	→	Server decrypts client's key exchange with its private key.
Client creates Change Cipher message; sends to server	→	
Client calculates symmetrical key, creates Finish message and encrypts with symmetrical key; sends to server	→	Server decrypts the Finish message received from client.
	←	Server creates Change Cipher message; sends to client
Client decrypts Finish message received from server	←	Server calculates symmetrical key, creates Finish message and encrypts with symmetrical key; sends to client.

Figure 8-1 SSL handshake without client authentication

Figure 8-2 on page 138 illustrates the SSL handshake when client authentication has been requested by the server.

Client		Server
Client starts SSL handshake; sends list of supported ciphers	→	Server selects cipher
Client authenticates server's certificate and gets server's public key	←	Server sends its certificate to client, includes request for client's certificate.
Client creates key exchange message and encrypts with server's public key; sends to server along with client's certificate and verification.	→	Server validates/authenticates the client's certificate; decrypts client's key exchange with its own private key.
Client creates Change Cipher message; sends to server	→	
Client calculates symmetrical key, creates Finish message and encrypts with symmetrical key; sends to server	→	Server decrypts the Finish message received from client.
	←	Server creates Change Cipher message; sends to client
Client decrypts Finish message received from server	←	Server calculates symmetrical key, creates Finish message and encrypts with symmetrical key; sends to client.

Figure 8-2 SSL handshake with client authentication

8.2.3 Enabling SSL in the CICS TG

The CICS Transaction Gateway supports the JSSE implementation of SSL. JSSE, as supplied with the Java Software Development Kit (SDK), is the only supported SSL option. Two tools, iKeyman and keytool, are shipped in both the Java Runtime Environment (JRE) and SDK packages. The **`keytool`** command is issued from the command line, and iKeyman is a graphical tool. These tools are used to work with digital certificates, which are then stored in a keystore on the local file system.

Because the JSSE code is 100% Java, it is platform independent and there is no z/OS specific code. However, there are a couple of functions in other Java security components that provide z/OS specific capabilities that JSSE can use. These functions include the ability to take advantage of the hardware cryptographic devices that can be present on the z/OS platform along with storing certificates in RACF. The **`hwkeytool`** command can be used to create certificates compatible with the System z hardware cryptography. RACF administration commands support the generation, importing, exporting, and storing of certificates.

8.2.4 Securing IPIC connections

An Internet Protocol interconnectivity (IPIC) connection can be established between the Gateway daemon and the CICS server or directly to the CICS server from a Java application server when a local mode connection is being used.

IPIC connections enforce the following securities:

- Bind security to prevent an unauthorized client system from connecting to CICS
- Link security to restrict the resources that can be accessed over a connection to a CICS server
- User security to restrict the CICS resources that can be accessed by a user

Bind security

For IPIC, bind security is supported by the exchange of SSL client certificates. The following list shows important configurations that allow the CICS TG to connect successfully, and to prevent an unauthorized system from connecting:

- The **SEC** system initialization parameter in CICS must be set to `YES`.
- The TCPIPSERVICE definition in CICS must specify:
  - SSL(`CLIENTAUTH`) or SSL(`YES`).
  - CIPHERS (`cipher_suite_code_list`).
  - Optionally, CERTIFICATE (`X.509_certificate_label`). The named certificate is used as the server certificate. If `CERTIFICATE` is not specified, the default certificate defined in the key ring for the CICS region user ID is used.
- The following list shows what must be configured for the Gateway daemon:
  - Set the key ring parameters.
  - Set the **ssl** parameter in the `IPICSERVER` section of the configuration file to `Y`.
  - Optionally, if you want to limit the cipher suites that are enabled for the connection, set the **ciphersuites** parameter to a comma-separated list of cipher suites to use.
- Local mode configurations must define the CICS server name:

  `ssl://<server_name>:<port>`

The CICS TG and CICS TS information centers provide the detailed description of the definitions required to configure SSL for IPIC connections. The CICS TG information center also contains scenarios illustrating how to perform the configuration tasks for both CICS TG and the CICS server.

Link security

There are two ways that you can specify the link user for IPIC connections. You can use the `SECURITYNAME` attribute, or an SSL certificate in the IPCONN definition in CICS. The client's certificate is mapped by the external security manager to a specific user ID, which is defined as the link user. Therefore, you can specify separate link users depending on which certificate you are using.

To specify a link user, set `LINKAUTH` in the IPCONN definition in CICS to one of the following settings:

- `SECUSER` to use the user ID that is specified in the `SECURITYNAME` attribute to establish link security
- `CERTUSER` to use an SSL client certificate mapped to a user ID to establish link security

When specifying `CERTUSER`, the IPCONN resource must refer to a TCPIPSERVICE definition that is configured for SSL and client authentication. The certificate must be mapped in the external security manager to your chosen user ID.

User security

IPIC connections can be configured to enforce user security to restrict the CICS resources that can be accessed by a user. The level of user security checking is specified by setting the `USERAUTH` attribute in the IPCONN definition in CICS. The `USERAUTH` setting in the IPCONN definition is comparable to the `ATTACHSEC` setting on other connection definitions.

- If `USERAUTH=IDENTIFY` is specified, a user ID that is already verified must be supplied. If the connecting CICS TG for z/OS component (a Gateway daemon or an application using local mode) and the CICS TS for z/OS region are not in the same Parallel Sysplex, a client-authenticated SSL connection is required.
- If `USERAUTH=VERIFY` is specified, a user ID and password or password phrase must be supplied. If password phrases are to be used, the CICS server must support password phrases.

Two other user security options exist in which CICS does not accept a user ID or password from clients. In these cases, a predefined user is assigned to each task that is started. The following list shows those other security options:

- If `USERAUTH=LOCAL` is specified, all requests run under the link user ID or the default user ID if there is no link user ID.
- If `USERAUTH=DEFAULTUSER` is specified, all requests run under the default user.

Identity propagation can be used as an alternative to specifying a user ID.

8.2.5 Securing connections to the Gateway daemon

When using a remote mode topology, the connection between Java and .NET clients and the Gateway daemon can be secured using SSL. SSL is not available for clients using ECIv2. If the client will be flowing the user ID and password, an SSL connection is advised to protect the confidentiality of this information.

Note: The CICS TG Information Centers include scenarios to show how to configure the following options:

- SSL security on the Gateway daemon
- SSL server authentication and (optionally) SSL client authentication
- Sending an ECI request to the CICS server to check that the SSL connection works

For more information about these options, see the following scenarios:

- Scenario SC06 in the CICS TG for Multiplatforms Information Center:

  http://ibm.co/1tfM7v1

- Scenario SC05 in the CICS TG for z/OS Information Center:

  http://ibm.co/1zStHVw

Security exits

The CICS Transaction Gateway products include a feature for the manipulation, validation, or auditing of application requests between Java and JCA client applications and the Gateway daemon. The feature is known as the *security exits*. Although security exits can be used for reasons other than security, several examples of their use are listed:

- Compressing data over the wire (therefore obscuring)
- Rejecting requests based on request content or metadata

Note: Aside from being limited to Java and JCA applications, another limitation of the security exits is the requirement for the application to specify the name of the Java classes to be used. This places an affinity into the remote client application, and also makes it difficult to implement when the Gateway daemon needs to serve a mixed set of applications.

CICS TG includes a set of Java components that define a simple yet flexible model for implementing custom security exits. Two of these can be used independently of whether SSL is also used:

- `com.ibm.ctg.security.ServerSecurity`

  This interface is used for server-side security classes that do not require the exposure of SSL client certificates.

- `com.ibm.ctg.security.ClientSecurity`

  This interface is used for all client-side security classes.

Two additional classes are designed to work in conjunction with the SSL environment on the Gateway daemon:

- `com.ibm.ctg.security.JSSEServerSecurity`

  The JSSEServerSecurity interface is an alternative to the Server Security interface and works in conjunction with the SSL protocol. The interface allows access to a client certificate passed during the initial SSL handshake. The exposure of the client certificate depends on configuring the CICS TG to support client authentication.

- `com.ibm.ctg.util.RACFUserid`

  This class (only available in the CICS TG for z/OS) is used to map an X.509 Client Certificate to a RACF user ID. The certificate must already be associated with a valid RACF user ID or an error is returned from RACF.

The CICS TG products include sample programs to illustrate the use of the security exits. They are principally designed to compress the data stream between the client application and the Gateway daemon:

- ClientCompression implements `ClientSecurity` and demonstrates data compression.
- ServerCompression implements `ServerSecurity` and demonstrates data compression.
- SSLServerCompression implements `JSSEServerSecurity` and demonstrates how to expose an SSL client certificate.

The source for these samples is in the following directory:

`<install_path>/samples/java/com/ibm/ctg/samples/security`

8.3 Passwords, PassTickets, and password phrases

During authentication, the identity of a user can be established by supplying both the user ID and a password. A password is typically a short sequence of letters, numbers, and certain special characters. On z/OS, a password can be up to eight characters long, and it might not be limited to uppercase, depending on the system configuration.

Note: Support for mixed-case passwords was introduced with CICS TG for z/OS V7.0.

Password phrases differ from traditional passwords in that they are longer than passwords, ranging from 9 to 100 characters long. This greater length is thought to provide more powerful security. Otherwise, they operate on the same principle as passwords, and they are used in exactly the same way. Not all CICS systems or CICS TG releases support the use of password phrases.

Note: Password phrase support was introduced by the CICS TG V8.1 products, and it can be used when connecting to CICS TS for z/OS V4.2, or later, using an IPIC connection. CICS TG for z/OS V9.0 added further support, allowing authentication of password phrases for external CICS interface (EXCI) connections and any supported release of CICS TS for z/OS.

A PassTicket is a program-generated character string to use directly instead of a password or password phrase. A PassTicket looks very much like a password, and it can be used by CICS TG applications directly in place of a normal password. However, PassTickets are only valid for a single authentication operation, and they expire within a short time after generation. The basis of the technology relies on the use of a shared secret key.

Using PassTickets removes the need to transmit reusable passwords, and password phrases, over the network. It also makes it possible for client systems to authenticate themselves to the server, without the need to securely manage reusable credentials in the client system.

For more information about PassTickets, see the topic, *The RACF PassTicket*, in the z/OS Information Center:

http://ibm.co/UgHsga

8.4 Identity propagation

Identity propagation is the capability where a non-z/OS identity (a distributed identity) is propagated into the z/OS environment and then is used for these functions:

- To provide credentials for authorization by being mapped to an existing security user ID
- To be available throughout the z/OS sysplex for auditing and reporting

The distributed identity of an authenticated user is determined by the distributed application and passed along to the receiving subsystem on z/OS. The subsystem, such as CICS, then uses the ESM (such as RACF) to search for a mapping between the distributed identity and a SAF user ID. The logon is failed if no mapping is found.

The distributed identity is then associated with the SAF user ID within the security context for the specific request. Both the SAF user ID and the distributed identity will now be included in any audit records that relate to the processing of the request.

Figure 8-3 on page 143 shows a conceptual overview of identity propagation.

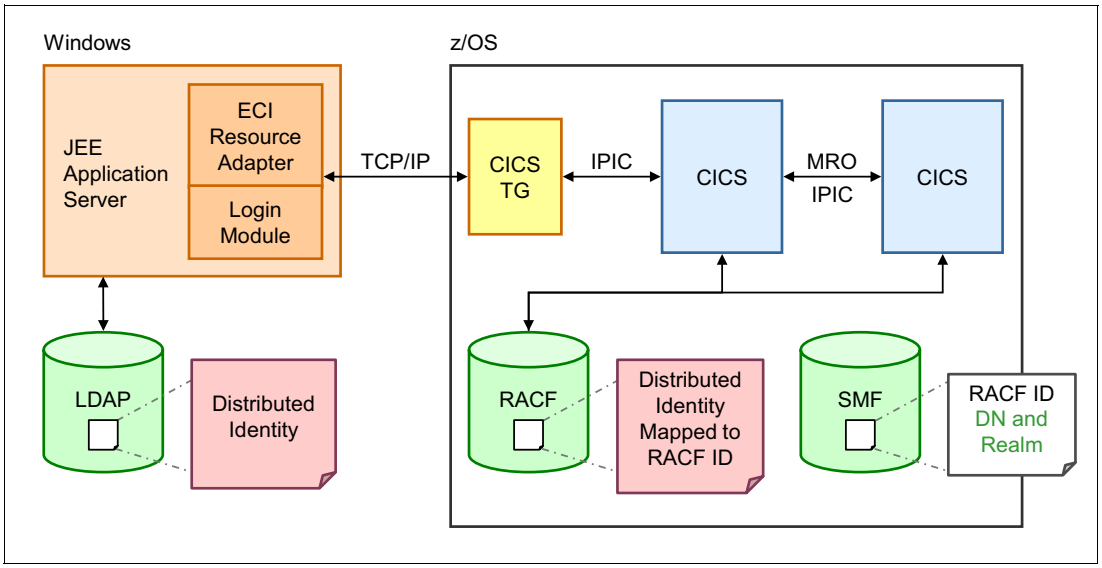


Figure 8-3 Identity propagation

One requirement for identity propagation is that the connection between the CICS TG and CICS must be an IPIC connection. A client-authenticated SSL connection is required unless CICS Transaction Gateway and CICS Transaction Server are on z/OS and in the same sysplex.

In the information center for CICS TG for z/OS, Scenario SC04 shows how to configure CICS TG for z/OS, CICS TS, RACF, and WebSphere Application Server on AIX. Specifically, this scenario shows how to implement identity propagation of a user's security information (the distributed identity), which is held in IBM Tivoli Directory Server. When this is passed to CICS Transaction Server, the identity is mapped to a user ID in RACF.

For a more thorough description of identity propagation (ID propagation) and detailed examples for implementation, see *z/OS Identity Propagation*, SG24-7850:

http://www.redbooks.ibm.com/abstracts/sg247850.html

8.5 The external security interface

The external security interface (ESI) provides a security management API to manage user IDs and passwords. ESI enables user applications to perform security-related tasks, such as viewing the status of user IDs and updating passwords held by an ESM.

The original implementation of ESI was an interface to Password Expiration Management (PEM), and part of the Advanced Program-to-Program Communication (APPC) architecture. Its use was restricted to CICS TG on distributed platforms (CICS TG for Multiplatforms or the now discontinued CICS Universal Client), and then only when the gateway products connected to CICS using Systems Network Architecture (SNA). As of CICS TG V8.1, ESI is supported over IPIC connections, therefore available to all CICS TG products.

ESI can be used to verify and change the user ID and password information stored in the ESM. Two primary functions provided by ESI are verifying passwords and changing passwords. These invoke the corresponding CICS commands **`EXEC CICS VERIFY PASSWORD`** and **`EXEC CICS CHANGE PASSWORD`**. These commands interface with the ESM, such as RACF, to perform the operation that you want.

Part 3

Configuration

This part shows how to configure connections in CICS, configure simple end-to-end connectivity (all platforms) with verification for remote and local mode applications, and add security, extended architecture (XA) support, and high availability.

9

Configuring IPIC in CICS Transaction Gateway for z/OS

This chapter describes the configuration of IPIC connections in CICS Transaction Server for z/OS (CICS TS), used by both the CICS TG for z/OS and CICS TG for Multiplatforms scenarios in this book, including the sample CICS programs we used for validation. Using IPIC delivers the most feature-rich set of capabilities for CICS TG, and IPIC is available for use by all of the CICS TG products.

This chapter explains how to secure IPIC connections using Secure Sockets Layer (SSL) server authentication, and how an alternative IPIC autoinstall user exit is used to handle multiple connections originating from a WebSphere Application Server cluster.

For an overview of the CICS connection protocols and several of the issues related to using autoinstall with IPIC, see Chapter 4, “Connecting to CICS” on page 43.

This chapter covers the following topics:

9.1 Configuration overview

In this chapter, we will describe the customizations made to our CICS TS for z/OS region, `TOR1`, required to establish and verify IPIC connectivity with a CICS TG for z/OS installation, and a CICS TG for Multiplatforms installation. The CICS TS definitions described are consistent with the CICS TG definitions described in Chapter 10, “CICS Transaction Gateway for z/OS configuration” on page 171 and Chapter 11, “CICS Transaction Gateway Multiplatforms and Desktop Edition configuration” on page 205.

Note: In this chapter, unqualified references to “CICS TS” refer to the product, CICS Transaction Server for z/OS.

Figure 9-1 shows the topology of the basic system described through these three chapters. This topology forms the basis of the High Availability topology described in Chapter 13, “High availability configuration” on page 295.

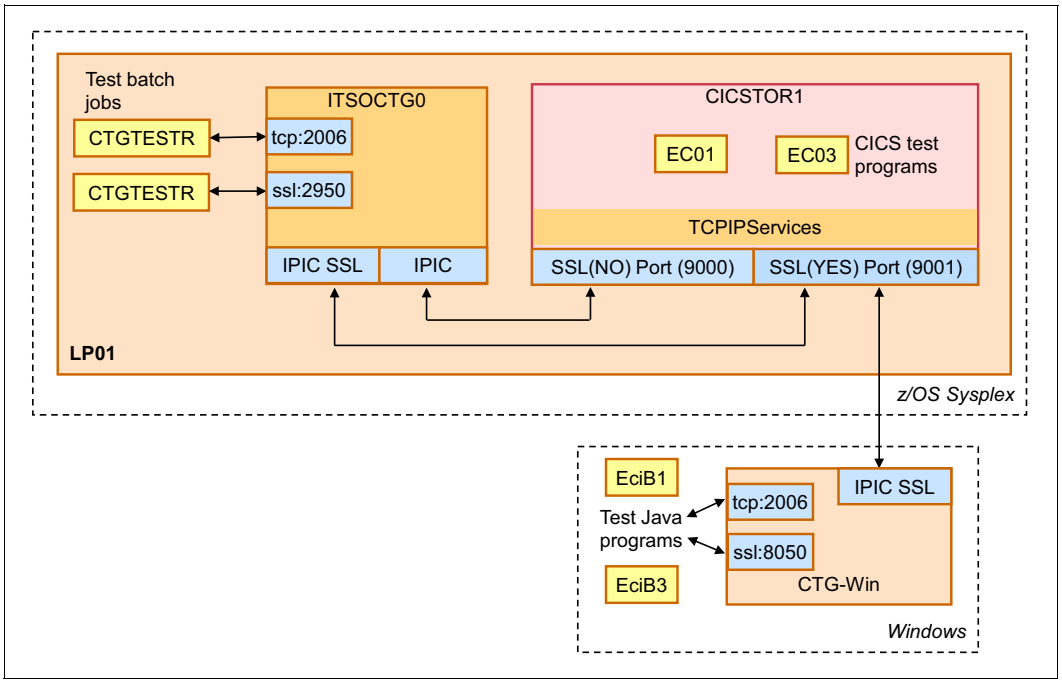


Figure 9-1 The basic system topology implemented in Part 3 of the book

The basic CICS TS configuration includes a single Terminal/Transport owning region (TOR) with IPIC connectivity allowing a Gateway daemon to connect to the CICS TS region, and two CICS programs installed in the TOR, representing the business logic to be invoked through CICS TG. The IPIC connection is secured using SSL server authentication.

Note: The CICS Transaction Gateway product Information Centers contain a scenario, *Configuring SSL security between a Java Client and the Gateway daemon*, Client Authentication.

A typical CICS TS configuration uses an Application Owning Region (AOR) to separate the target business logic from the TOR regions. However, to allow this book to focus on the CICS TG concepts, a single combined TOR/AOR region is used.

The basic CICS TS configuration is designed to handle up to 50 concurrent ECI requests. Of course, this is a relatively low number, but the aim of the scenario is to start with a relatively simple end-to-end solution, and then demonstrate scalability and availability through Chapter 13, “High availability configuration” on page 295.

Table 9-1 summarizes the CICS TS configuration settings for this chapter.

Table 9-1 CICS TOR configuration summary

Configuration parameter	Value
CICS APPLID	CICSTOR1
CICS region User ID	CICSRS3
CICS KEYRING	Cics.Cicstor1
TCPIPSERVICE port for IPIC, no SSL	9000
TCPIPSERVICE port for IPIC SSL	9001
TCPIPSERVICE certificate for IPIC SSL	Cicstor1-IPIConnection
TCPIPSERVICE host for logical partition (LPAR) LP01	lp01.redbooks.ibm.com

9.2 CICS definitions for IPIC

To establish an IPIC connection to a CICS server, the following components must be in place:

- TCP/IP support must be active in the CICS region.
- There must be an installed TCPIPSERVICE resource, using the IPIC protocol.
- The CICS TS region must contain an installed IPCONN resource.

  The IPCONN resource can be derived from an explicit IPCONN resource definition, or through the autoinstall mechanism.

This section describes the configuration changes made to the `CICSTOR1` region, providing non-secure IPIC connections for use by the Gateway daemon configurations described in Chapter 10, “CICS Transaction Gateway for z/OS configuration” on page 171 and Chapter 11, “CICS Transaction Gateway Multiplatforms and Desktop Edition configuration” on page 205. We used predefined IPCONN resources rather than autoinstall for the basic system, because they allow you to directly associate definitions between the chapters.

Note: Securing IPIC connections using SSL is described in 9.4, “Securing IPIC with SSL” on page 162, which builds on the steps described in this section. It is therefore suggested to read both sections when implementing secure IPIC connectivity.

CICS TS configuration for autoinstall of IPIC connections is covered in 9.5, “Configuring IPIC autoinstall for WebSphere Application Server clusters” on page 167, and it is required by the high availability configuration described in Chapter 13, “High availability configuration” on page 295.

9.2.1 System initialization parameters

A CICS region is enabled for TCP/IP-based communication services, such as IPIC, or ECI over TCP/IP connections by setting the **TCPIP** system initialization parameter to YES. We set this system initialization parameter in our CICS region using the system initialization table (SIT) override option:

```
TCPIP=YES
```

Note: The default value for the TCPIP system initialization parameter is NO.

Confirm that TCPIP services are available to the CICS region using the command:

```
CEMT INQUIRE TCPIP
```

The output from the CEMT command indicates that TCPIP is open, as shown in Figure 9-2.

```
I TCPIP
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Tcp Ope Act(00001) Max( 00255 ) Cic
```

Figure 9-2 Verifying TCPIP services are active in the CICS region

The ISC system initialization parameter

The CICS Transaction Server for z/OS Information Center states that the **ISC** system initialization parameter must be set to YES to use IPIC connections. We found that this is not necessary when using IPIC exclusively with CICS Transaction Gateway components (a Gateway daemon, or a local mode application).

Because a combined TOR/AOR CICS region is used in the basic system for this book, and the sample CICS programs do not use remote resources, setting the **ISC** parameter to YES is not strictly necessary. However, such simple configurations are uncommon except within development or test environments, and for production environments, the **ISC** parameter typically must be set to YES.

Note: When a CICS program issues a dynamic program link (DPL) for a remote system from a CICS region with **ISC** set to NO, an ABEND AEY9 is issued.

9.2.2 The mirror transaction

For a CICS region to process an ECI request from CICS Transaction Gateway, a "*mirror transaction*" must be defined. A mirror transaction issues a DPL request on behalf of the ECI application, invoking the CICS program specified by the ECI request.

By default, the transaction ID of the mirror is either CSMI, or CPMI. When using the CICS Transaction Gateway for z/OS, the default is CSMI. When using the CICS Transaction Gateway for Multiplatforms, or CICS Transaction Gateway Desktop Edition, the default is CPMI.

The default mirror transaction ID can be overridden in the ECI application by setting a transaction program name (TPN) name, or through the **TPName** attribute for a JCA connection factory (in a WebSphere Application Server). A non-default mirror transaction ID can be used to demarcate separate lines of business for routing requests, to distinguish levels of resource access within business logic by the invoking application, or simply for monitoring purposes.

In this book, only the default mirror transactions `CSMI` and `CPMI` are used. Definitions for both of these transactions are included in the CICS-supplied Resource Definition Online (RDO) group, `DFHISC`. This group is installed by issuing the following command from a CICS terminal:

```
CEDA INSTALL GROUP(DFHISC)
```

> **Note:** The CICS-supplied RDO group `DFHISC` also includes the PROGRAM definition for the COMMAREA conversation table, DFHCNV, described in 9.3.3, “Data conversion for the COMMAREA samples” on page 160.

9.2.3 TCPIPSERVICE resource definition

This section describes the definition of a “listener port” in CICS, a well-known entry point for non-secure IPIC connection requests from partner systems, such as CICS Transaction Gateway.

On the `CICSTOR1` region, create a TCPIPSERVICE definition `GW1IPIC` in group `T1IPIC` by issuing the following command from a CICS terminal:

```
CEDA DEFINE TCPIPSERVICE(GW1IPIC) GROUP(T1IPIC)
```

Accept the default values for all parameters, except for the following parameters:

- **`Urm`**: `NO`

  Specifying `NO` disables autoinstall for IPIC partners connecting to this TCPIPSERVICE.

- **`POrtnumber`**: `9000`

  The port on which TOR is listening for inbound IPIC connection requests.

- **`PROtocol`**: `IPIC`

  The protocol to be used by partner systems connecting through this service.

- **`Backlog`**: `00010`

  Allow a sensible number of partner systems to queue for connection. Consider the number of independent IPIC connections likely to be established through this TCPIPSERVICE.

  > **Note:** Backlog is limited by the system default value, `somaxconn`. CICS TS for z/OS V5.1, APAR PM85188, brings many improvements to IPIC connectivity, including a change to the meaning of zero for the TCPIPSERVICE attribute, `Backlog`. A zero value now indicates that the system default value, `somaxconn`, needs to be used.

- **`Host`**: `lp01.redbooks.ibm.com`

  Binding the listener port to the unadvertised static virtual IP address (VIPA) on logical partition (LPAR) LP01 limits the likelihood of unwanted connections and unintentionally sharing the IPIC connection. For further details, see “Binding of listener ports” on page 300.

  > **Note:** The `Ipaddress` attribute of the TCPIPSERVICE is superseded by the `Host` attribute, which adds IPv6 support.

The full TCPIPSERVICE definition is shown in Example 9-1.

Example 9-1 TCPIPSERVICE definition for IPIC connections defined in TOR

```
TCpipservice   : GW1IPIC
 GROup          : T1IPIC
 DEScription    : IPIC connection for CICS TG
 Urm            : NO
 POrtnumber     : 09000
 STatus         : Open
 PROtocol       : IPic
 TRansaction    : CISS
 Backlog        : 00010
 TSqprefix      :
 Host           : lp01.redbooks.ibm.com
 (Mixed Case)   :
 Ipaddress      :
 SOcketclose    : No
 MAXPersist     : No
 MAXDatalen     : 000032
SECURITY
 SSl            : No
 CErtificate    :
 (Mixed Case)
 PRIvacy        :
 CIphers        :
 AUthenticate   :
 Realm          :
 (Mixed Case)
 ATtachsec      :
DNS CONNECTION BALANCING
 DNsgroup       :
 GRPcritical    : No
```

The TCPIPSERVICE definition was then installed by issuing the following command from a CICS terminal:

```
CEDA INSTALL TCPIPSERVICE(GW1IPIC) GROUP(T1IPIC)
```

The CICS log messages shown in Example 9-2 indicates that the TCPIPSERVICE resource was successfully installed and opened.

Example 9-2 CICSTOR1 region log messages for IPIC connection GW1IPIC

```
DFHSO0133 10/29/2013 13:12:25 CICSTOR1 TCPIPSERVICE GW1IPIC has been installed.
DFHSO0107 10/29/2013 13:12:25 CICSTOR1 TCPIPSERVICE GW1IPIC has been opened on
   port 09000 at IP address 192.168.0.1.
```

With the TCPIPSERVICE installed, the next step is to consider the IPCONN resources, which represent specific IPIC connections between partner systems and the CICS region.

Note: Despite defining the TCPIPSERVICE using the domain name server (DNS) host name, `lp01.redbooks.ibm.com`, the DFHSO0107 message shows the IP address.

9.2.4 IPCONN resource definition

This section describes the steps required to define, install, and verify IPCONN resources that are dedicated for use by specific partner systems. In this case, the specific partner systems are Gateway daemons. One z/OS Gateway daemon is described in Chapter 10, “CICS Transaction Gateway for z/OS configuration” on page 171, and another Multiplatforms Gateway daemon is described in Chapter 11, “CICS Transaction Gateway Multiplatforms and Desktop Edition configuration” on page 205.

Define the IPCONN for the Gateway daemon on z/OS

On the TOR, create an IPCONN definition `CTG1` in group `T1IPIC` by issuing the following command from a CICS terminal:

```
CEDA DEF IPCONN(CTG1) GROUP(T1IPIC)
```

Set the following attribute values as shown in Example 9-3:

- **APplid**: `ITSOCTGO`

  Match the **Applid** value specified in the configuration file `PRODUCT` section of the connecting Gateway daemon.

- **Networkid**: `CTGALPOO`

  Match the **ApplidQualifier** value specified in the configuration file `PRODUCT` section of the connecting Gateway daemon.

- **Tcpipservice**: `GW1IPIC`

  The name of the TCPIPSERVICE resource definition defined previously.

- **Receivecount**: `50`

  The maximum number of logical sessions available for this IPCONN resource, allowing up to 50 concurrent application requests over this connection.

> **Note:** The **Receivecount** attribute limits the possible number of concurrent requests an IPCONN can accept, and it is directly related to the **SendSessions** attribute of the CICS TG IPICSERVER definition. The actual limit can be negotiated to a lower value by the partner CICS TG component during connection establishment. It cannot be negotiated to a higher value.

Example 9-3 Predefined IPCONN definition in TOR

```
Ipconn         : CTG1
 Group          : T1IPIC
 DEScription    : IPIC definition for CICS TG
IPIC CONNECTION IDENTIFIERS
 APplid         : ITSOCTGO
 Networkid      : CTGALPOO
 Host           :
 (Mixed Case)   :
 Port           : No
 Tcpipservice   : GW1IPIC
IPIC CONNECTION PROPERTIES
 Receivecount   : 050
 SENdcount      : 000
 Queuelimit     : No
 MAxqtime       : No
OPERATIONAL PROPERTIES
```

```
   AUtoconnect    : No
  INservice      : Yes
 SECURITY
  SSl            : No
  CErtificate    :
  CIphers        :
  Linkauth       : Secuser
  SECurityname   :
  Userauth       : Local
  IDprop         : Notallowed
 RECOVERY
  Xlnaction      : Keep
 MIRROR TASK PROPERTIES
  MIrrorlife     : Request
```

The IPCONN definition is installed by issuing the following command from a CICS terminal:

CEDA INSTALL IPCONN(CTG1) GROUP(T1IPIC)

CICS log message DFHIS3030 indicates that the IPCONN resource was successfully installed:

```
DFHIS3030 I 11/08/2013 09:40:45 CICSTOR1 IPCONN CTG1 installed.
```

Further verify the status of the IPCONN resource by using the following command from the CICS terminal:

CEMT INQUIRE IPCONN(CTG1)

The expanded output from the **CEMT** command is shown in Example 9-4.

Example 9-4 The full output from the CEMT INQUIRE command for an IPCONN resource

```
I IPCONN(CTG1)
 RESULT - OVERTYPE TO MODIFY
   Ipconn(CTG1)
   Applid(CTG1ITSO)
   Networkid(CTGALPOO)
   Servstatus( Inservice )
   Connstatus( Released )
   Ssltype(Nossl)
   Purgetype(              )
   Receivecount(050)
   Sendcount(000)
   Tcpipservice(GW1IPIC)
   Port()
   Host()
   Hosttype()
   Ipresolved(0.0.0.0)
   Ipfamily(Unknown)
   Pendstatus( Notpending )
   Recovstatus( Norecovdata )
   Uowaction(              )
   Uowaction(              )
   Queuelimit()
   Maxqtime()
   Userauth(Identify)
   Linkauth(Secuser)
   Idprop(Notallowed)
```

```
Securityname()
Certificate()
Ciphers()
Partner()
Mirrorlife(Request)
```

The **Constatus** attribute has a value of Released until the specific partner system, the z/OS Gateway daemon with the Gateway ID CTGALP00.CTG1ITS0 establishes the IPIC connection in 10.2.2, “Test using the CTGTESTR job” on page 181.

Defining the IPCONN for the Gateway daemon on Multiplatforms

Creating the IPCONN resource definition to connect with the Gateway daemon on Windows, which is described in Chapter 11, follows the same procedure as “Define the IPCONN for the Gateway daemon on z/OS” on page 153, replacing only the following attributes and values:

- **Applid**(CTGMWIN1)
- **Networkid**(CTGMP)

11.5.4, “Secure the IPIC connection” on page 248 adds a secure IPIC connection from the Gateway daemon on Multiplatforms and therefore requires an additional IPCONN definition using the TCPIPSERVICE definition GW1IPICS, described by 9.4, “Securing IPIC with SSL” on page 162.

Verifying the TCPIPSERVICE attributes

There are many tuning parameters in TCP/IP that might have a dramatic effect on the performance of the overall solution. Some of these parameters overlap with CICS resource definitions, such as the Backlog attribute of the TCPIPSERVICE resource. A conflict might occur when the system-wide maximum (somaxconn) is lower than the value specified in the TCPIPSERVICE definition, because CICS continues to show that the higher value is in effect.

The **netstat** command shows the true value that is in effect, with the MAXIMUMBACKLOG attribute.

The **netstat** command (either the TSO or UNIX System Services version) can provide extensive information about open TCP/IP listener ports, and active connections, including the true backlog value (the MAXIMUMBACKLOG attribute). The following TSO command provides detailed output regarding all TCP/IP resources owned by the CICS region, CICSTOR1:

```
d tcpip,tcpip,netstat,all,cli=cicstor1
```

Example 9-5 shows an extract of the **netstat** command output related to the TCPIPSERVICE GW1IPIC, listening on port 9000.

Example 9-5 Output from the TSO netstat command for a specific CICS region

```
CLIENT NAME: CICSTOR1                  CLIENT ID: 00004792
LCLSOCK: 192.168.0.1..9000             FGNSOCK: 0.0.0.0..0
  BYTESIN:            0000000000         BYTESOUT:           0000000000
  SEGMENTSIN:         0000000000         SEGMENTSOUT:        0000000000
  STARTDATE:          03/25/2014         STARTTIME:          16:49:08
  LAST TOUCHED:       16:49:08           STATE:              LISTEN
  RCVNXT:             0000000000         SNDNXT:             0000000000
  CLIENTRCVNXT:       0000000000         CLIENTSNDNXT:       0000000000
  INITRCVSEQNUM:      0000000000         INITSNDSEQNUM:      0000000000
  CONGESTIONWINDOW:   0000000000         SLOWSTARTTHRESHOLD: 0000000000
  INCOMINGWINDOWNUM:  0000000000         OUTGOINGWINDOWNUM:  0000000000
  SNDWL1:             0000000000         SNDWL2:             0000000000
```

```
SNDWND:              0000000000        MAXSNDWND:          0000000000
SNDUNA:              0000000000        RTT_SEQ:            0000000000
MAXIMUMSEGMENTSIZE: 0000000536         DSFIELD:            00
ROUND-TRIP INFORMATION:
  SMOOTH TRIP TIME: 0.000              SMOOTHTRIPVARIANCE: 1500.000
REXMT:               0000000000        REXMTCOUNT:         0000000000
DUPACKS:             0000000000        RCVWND:             0000131072
SOCKOPT:             80                TCPTIMER:           00
TCPSIG:              00                TCPSEL:             00
TCPDET:              C0                TCPPOL:             00
TCPPRF:              00
QOSPOLICY:           NO
ROUTINGPOLICY:       NO
RECEIVEBUFFERSIZE:   0000065536        SENDBUFFERSIZE:     0000065536
CONNECTIONSIN:       0000000000        CONNECTIONSDROPPED: 0000000000
MAXIMUMBACKLOG:      0000000010        CONNECTIONFLOOD:    NO
CURRENTBACKLOG:      0000000000
  SERVERBACKLOG:     0000000000        FRCABACKLOG:        0000000000
CURRENTCONNECTIONS: 0000000000         SEF:                100
QUIESCED:            NO
APPLICATION DATA:    DFHICICSTOR1CISSIPIC    TEST
```

The `APPLICATION DATA` field is a useful cross-check, showing that port `9000` is owned by `CICSTOR1`, the associated TCPIPSERVICE resource is defined to use transaction `CISS`, and the protocol is `IPIC`. The send and receive buffer sizes, which are indicated by attributes `RECEIVEBUFFERSIZE` and `SENDBUFFERSIZE`, are interesting also. They need to be set to at least 64 KB for optimum efficiency.

9.3 The sample application programs

This section describes the steps taken to install and configure CICS TS to use the CICS COBOL sample programs that are provided in the CICS Transaction Gateway products.

CICS TG products include client application samples in multiple programming languages (Java, Java EE, C, C++, C#, and Visual Basic), and also CICS program samples in COBOL. For CICS Transaction Gateway for Multiplatforms and CICS Transaction Gateway Desktop Edition, these samples are in the product installation `samples` directory. For CICS Transaction Gateway for z/OS, the `samples` directory is in the product installation z/OS UNIX file system.

Within the `samples` directory, the `server` subdirectory contains the following source code for CICS COBOL programs:

- `ec01.ccp`: A COMMAREA program that returns a formatted time/date string
- `ec02.ccp`: A COMMAREA program that returns a simple run counter
- `ec03.ccp`: A channel program that takes an input data container and returns three containers:
  - A data/time container
  - The length of the input data and returns uses channels and containers in a CICS program
  - An output container that contains a copy of the input data, or an error message

These are relatively simple CICS programs, which are designed to be invoked by the various CICS TG ECI sample programs. In this book, we used the `ec01.ccp` and `ec03.ccp` programs.

To make these CICS COBOL programs available to our CICS systems, the source code must be transferred to a z/OS partitioned data set (PDS), compiled, linked, defined, and installed to CICS.

9.3.1 Transferring, compiling, and linking the CICS sample programs

To translate, compile, and link the CICS sample programs, they must be copied into an MVS data set or PDS. For convenience, we used a `CICSRS3.CICS.SOURCE` PDS.

Because we installed the CICS Transaction Gateway for z/OS on the same LPAR as CICS TS, we can simply copy the COBOL sample programs directly from the z/OS UNIX file system to our PDS with a single TSO command:

```
OGETX '/usr/lpp/cicstg/ctg900/samples/server' 'CICSRS3.CICS.SOURCE' LC SUFFIX(ccp)
TEXT
```

> **Note:** Issue the **`OGETX`** command from the Interactive System Productivity Facility (ISPF) command processor panel (typically, option 6 on the ISPF menu). Option 6 is usually preferable, because it does not convert the commands that you enter into uppercase.

Users with CICS Transaction Gateway for Multiplatforms or CICS Transaction Gateway Desktop Edition use an appropriate FTP program in ASCII transfer mode to copy the CICS sample programs to an MVS data set.

The next step translates, compiles, and links the CICS sample COBOL programs in a single batch JOB, `CTGECIS`, that is shown in Example 9-6.

Example 9-6 Batch job to translate, compile, and link the CICS sample COBOL programs

```
//CTGECIS JOB CLASS=E,MSGCLASS=A,MSGLEVEL=(2,0)
/*JOBPARM SYSAFF=LP01
//* ---------------------------------------------------------
//EC01   EXEC DFHYITVL,TRNPARM='COBOL2,CICS,SP',
//          PROGLIB='CICSRS3.CICS.LOAD',
//          INDEX='CICSTS51.CICS',OUTC='*'
//TRN.SYSIN  DD DISP=SHR,DSN=CICSRS3.CICS.SOURCE(EC01)
//LKED.SYSIN DD *
   NAME EC01(R)
/*
//*
//EC02   EXEC DFHYITVL,TRNPARM='COBOL2,CICS,SP',
//          PROGLIB='CICSRS3.CICS.LOAD',
//          INDEX='CICSTS51.CICS',OUTC='*'
//TRN.SYSIN  DD DISP=SHR,DSN=CICSRS3.CICS.SOURCE(EC02)
//LKED.SYSIN DD *
   NAME EC02(R)
/*
//*
//EC03   EXEC DFHYITVL,TRNPARM='COBOL2,CICS,SP',
//          PROGLIB='CICSRS3.CICS.LOAD',
//          INDEX='CICSTS51.CICS',OUTC='*'
//TRN.SYSIN  DD DISP=SHR,DSN=CICSRS3.CICS.SOURCE(EC03)
//LKED.SYSIN DD *
```

```
   NAME EC03(R)
/*
//*
```

After submitting the batch job CTGECIS, the CICS sample COBOL programs are translated by the CICS translator, compiled by the COBOL compiler, and linked to the load library CICSRS3.CICS.LOAD. The CICS-supplied procedure DFHYITVL produces a significant amount of output, but the return code summary provides a good indication of the outcome, as shown in Example 9-7.

Example 9-7 Extract from the CTGECIS batch job output

```
11.27.01 JOB10894  IRR010I  USERID CICSSYSP IS ASSIGNED TO THIS JOB.
11.27.02 JOB10894  ICH70001I CICSSYSP LAST ACCESS AT 11:26:59 ON MONDAY, OCTO
11.27.02 JOB10894  $HASP373 CTGECIS  STARTED - INIT F    - CLASS E - SYS LP01
11.27.02 JOB10894  -ASID     0043     SC66                          --TIMINGS
11.27.02 JOB10894  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP   CONN    TCB
11.27.02 JOB10894  -CTGECIS           EC01        00    137     20 551177
11.27.02 JOB10894  -CTGECIS           EC01        00    484     94 551177
11.27.02 JOB10894  -CTGECIS           EC01        00     45      8 551177
11.27.02 JOB10894  -CTGECIS           EC01        00    200     43 551177
11.27.02 JOB10894  -CTGECIS           EC02        00    141     18 551177
11.27.03 JOB10894  -CTGECIS           EC02        00    494     94 551177
11.27.03 JOB10894  -CTGECIS           EC02        00     40      6 551177
11.27.03 JOB10894  -CTGECIS           EC02        00    197     43 551177
11.27.03 JOB10894  -CTGECIS           EC03        00    175     20 551177
11.27.03 JOB10894  -CTGECIS           EC03        00    560    100 551177
11.27.03 JOB10894  -CTGECIS           EC03        00     41      7 551177
11.27.03 JOB10894  -CTGECIS           EC03        00    204     44 551177
```

For more information about the CICS-supplied procedure DFHYITVL, see the CICS TS for z/OS Information Center topic, *Using the CICS-supplied procedures to install application programs*:

http://ibm.co/UgHzID

9.3.2 Defining and installing the CICS sample program

This section describes the CICS configuration steps required to define and install the CICS PROGRAM resources associated with the CICS TG-supplied sample programs, and how to create a DFHCNV table to perform data conversion for the COMMAREA-based samples, EC01 and EC02.

PROGRAM definitions for the samples

On the CICSTOR1 region, create a PROGRAM definition EC01 in group TORCTG by issuing the following command from a CICS terminal:

```
CEDA DEFINE PROGRAM(GW1IPIC) GROUP(TORCTG)
```

Set the following attribute values as shown in Example 9-8 on page 159:

- **DEScription**: CICS TG EC01 server sample
- **Language**: COBOL
- **DAtalocation**: Any

The full PROGRAM definition for EC01 is shown in Example 9-8 on page 159.

Example 9-8 Program definition for EC01

```
PROGram        : EC01
 Group          : TORCTG
 DEScription    : CICS TG EC01 server sample
 Language       : COBOL
 RELoad         : No
 RESident       : No
 USAge          : Normal
 USElpacopy     : No
 Status         : Enabled
 RSl            : 00
 CEdf           : Yes
 DAtalocation   : Any
 EXECKey        : User
 COncurrency    : Threadsafe
 Api            : Cicsapi
REMOTE ATTRIBUTES
 DYnamic        : No
 Status         : Enabled
 RSl            : 00
 CEdf           : Yes
 DAtalocation   : Any
 EXECKey        : User
 COncurrency    : Threadsafe
 Api            : Cicsapi
REMOTE ATTRIBUTES
 DYnamic        : No
 REMOTESystem   :
 REMOTEName     :
 Transid        :
 EXECUtionset   : Dplsubset
JVM ATTRIBUTES
 JVM            : No
 JVMClass       :
 REMOTESystem   :
 REMOTEName     :
 Transid        :
 EXECUtionset   : Dplsubset
JVM ATTRIBUTES
 JVM            : No
 JVMClass       :
 JVMServer      :
 JVMProfile     :
JAVA PROGRAM OBJECT ATTRIBUTES
 Hotpool        : No
```

Example 9-8 was used as a template to define `EC02` and `EC03`, simply changing the PROGRAM and description fields accordingly.

9.3.3 Data conversion for the COMMAREA samples

Traditional CICS programs suitable for invocation through an ECI request or linking via Dynamic Program Link (DPL) use a well known convention to exchange data. This convention is known as a *COMMAREA*. For CICS, it consists of unstructured data with a maximum size of 32 KB. When a CICS COMMAREA program is invoked from a remote system, a requirement might exist to convert all, or part, of a COMMAREA between code pages. COMMAREA data conversion can be implemented at a system level through the use of the CICS conversion table known as DFHCNV.

> **Note:** CICS programs using channels and containers, such as the sample program `EC03`, handle data conversion at the application level, and operate independently of DFHCNV.

For a general overview of the data conversion process with CICS TS, see the CICS TS for z/OS Information Center topic, *The conversion process*:

http://ibm.co/1szxnth

Test programs `EC01` and `EC02` return EBCDIC text data by default, which is not ideal for client applications running on a distributed platform. We created a CICS DFHCNV table with entries for the `EC01` and `EC02` programs, enabling input and output data to be converted to ASCII code page 437, which is suitable for the client application, and EBCDIC code page 037, which is suitable for the CICS program.

Example 9-9 shows the source code required to assemble a DFHCNV table for CICS programs `EC01` and `EC02`. On our test system, it is stored in the `CICSRS3.CICS.SOURCE` PDS as member `ECICNV`.

Example 9-9 DFHCNV table entries for the CICS sample programs EC01 and EC02

```
DFHCNV   TITLE 'DFHCNV ASCII TO EBCDIC CONVERSION TABLE'
         DFHCNV TYPE=INITIAL,CDEPAGE=437
*
         DFHCNV TYPE=ENTRY,RTYPE=PC,RNAME=EC01
         DFHCNV TYPE=SELECT,OPTION=DEFAULT
         DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=17,      *
               LAST=YES
*
         DFHCNV TYPE=ENTRY,RTYPE=PC,RNAME=EC02
         DFHCNV TYPE=SELECT,OPTION=DEFAULT
         DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=39,      *
               LAST=YES
*
         DFHCNV TYPE=FINAL
         END
```

The `DFHCNV TYPE=INITIAL` macro defines the default client and server code pages to use for entries that omit specific code pages. Specifying the **`CDEPAGE`** parameter with the value `437` is equivalent to setting `CLINTCP=437, SRVERCP=037`.

The values for the **`OFFSET`** and **`DATALEN`** parameters of the DFCNV macro are determined by an in-depth knowledge of the COMMAREA data structure that is expected by each CICS program. In `EC01` and `EC02`, the entire COMMAREA is considered to be character data. Therefore, the `OFFSET` attribute is `0`, and the `DATALEN` attribute reflects the overall length.

However, actual COMMAREA structures typically contain a mixture of character and binary data. For conversions, more complex sequences of DFHCNV macro entries are required, applying code page conversion in specific zones of the COMMAREA.

The source code for each CICS program defines this structure as DFHCOMMAREA. In the CICS sample programs `EC01` and `EC02`, both have structured COMMAREA data ending with a single byte, whose value is set to `LOW-VALUES`. This final byte acts as a null-terminator, allowing the output to easily be processed as a string by a C program, and it does not need to be converted. Therefore, the values shown in Example 9-9 on page 160 are both one byte less than the size of the corresponding DFHCOMMAREA definitions.

Assemble and link-edit the DFHCNV table source code using the JCL job shown in Example 9-10. This step creates a DFHCNV load module in the same load library as the CICS sample program load modules, described in 9.3.1, “Transferring, compiling, and linking the CICS sample programs” on page 157.

Example 9-10 JCL job to assemble and link-edit a DFHCNV table

```
//ECICNV JOB CLASS=E,MSGCLASS=A,MSGLEVEL=(2,0)
/*JOBPARM SYSAFF=LP01
//*-------------------------------------------------------------------
// SET INPUT='CICSRS3.CICS.SOURCE'
// SET OUTPUT='CICSRS3.CICS.LOAD'
// SET INDEX='CICSTS51.CICS'
//ASSEM    EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT2   DD DSN=&&TEMPPDS(MACROS),DISP=(,PASS),
//            SPACE=(CYL,(1,1,5)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
//SYSUT1   DD DISP=SHR,DSN=&INPUT.(ECICNV)
//*-------------------------------------------------------------------
//ASM      EXEC PGM=ASMA90,REGION=2048K,
//            COND=(3,LT,ASSEM),
//            PARM=('SYSPARM(INITIAL)',DECK,NOOBJECT,ALIGN)
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DSN=&INDEX..SDFHMAC,DISP=SHR
//         DD DSN=&INDEX..SDFHSAMP,DISP=SHR
//         DD DSN=SYS1.MACLIB,DISP=SHR
//         DD DSN=SYS1.MODGEN,DISP=SHR
//SYSUT1   DD SPACE=(CYL,(1,1))
//SYSUT2   DD SPACE=(CYL,(1,1))
//SYSUT3   DD SPACE=(CYL,(1,1))
//SYSIN    DD DSN=&&TEMPPDS(MACROS),DISP=(OLD,PASS)
//SYSPUNCH DD DSN=&&OBJMOD,DISP=(,PASS),
//            SPACE=(CYL,(1,1)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
//*-------------------------------------------------------------------
//BLDMBR   EXEC PGM=IEBUPDTE,PARM=NEW,
//            COND=((3,LT,ASSEM),(7,LT,ASM))
//SYSPRINT DD DUMMY
//SYSUT2   DD DSN=&&TEMPPDS,DISP=(OLD,PASS)
//SYSIN    DD DSN=&&OBJMOD,DISP=(OLD,PASS)
//*-------------------------------------------------------------------
//LKED     EXEC PGM=IEWL,
//            PARM=(LIST,XREF,LET,NCAL),
```

```
//            COND=((3,LT,ASSEM),(7,LT,ASM),(3,LT,BLDMBR))
//SYSPRINT DD SYSOUT=*
//*SYSLIB   DD DSN=SYS1.TELCMLIB,DISP=SHR
//SYSPUNCH DD DSN=&&TEMPPDS,DISP=(OLD,PASS)
//SYSUT1   DD SPACE=(CYL,(1,1))
//SYSLMOD  DD DSN=&OUTPUT.(DFHCNV),DISP=SHR
//SYSLIN   DD DSN=&&TEMPPDS(LNKCTL),DISP=(OLD,PASS),
//            VOL=REF=*.ASSEM.SYSUT2
//*-----------------------------------------------------------------
//DELTEMP  EXEC PGM=IEFBR14
//DD1      DD DSN=&&OBJMOD,DISP=(OLD,DELETE)
//DD2      DD DSN=&&TEMPPDS,DISP=(OLD,DELETE)
```

There is no simple method to verify the details encapsulated within a DFHCNV load module, other than through a formatted dump of the CICS region, or a third-party utility. To append or change the conversion table, the load module is regenerated in its entirety from the amended macro source.

For more information about the DFHCNV macro, see the CICS TS for z/OS Information Center topic, *Using the CICS-supplied procedures to install application programs*:

http://ibm.co/1uOTQRI

9.4 Securing IPIC with SSL

To demonstrate end-to-end secure connectivity, we use self-signed certificates. The CICS sample REXX script **SDFHSAMP(DFH$RING)** was customized to create a new key ring, add a CERTAUTH certificate, and add a personal certificate for a CICS region to the new key ring.

The certificate authority (CA) certificate was used to sign personal certificates for the Gateway daemon and client applications. This section describes the steps to create these resources, and how they were used to configure a secure IPIC connection in the CICS region CICSTOR1. Figure 9-3 on page 163 shows the basic system configuration, now including details about the components that use RACF key rings or Java keystores.

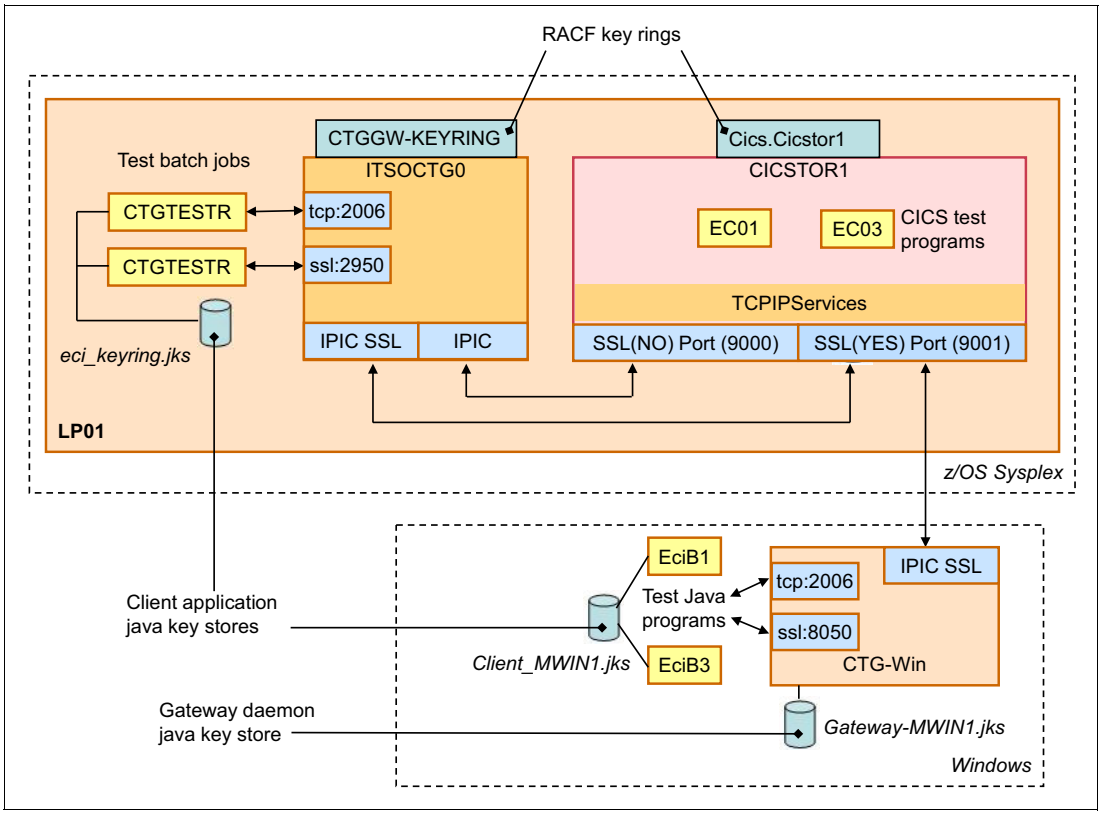


Figure 9-3 The basic system topology that includes details of RACF key rings and Java keystores

Each component in the system requiring secure connectivity requires a personal certificate to identify itself. Because the z/OS Gateway daemon and CICS region `CICSTOR1` have personal certificates signed by our signer CA, the signer CA must be imported into the key ring, or keystore, of each connected system. It must also be marked as trusted, which is indicated by an entry type of `trustedCertEntry`, when using **`keytool`**.

The result is that each Gateway daemon trusts the personal certificate of the CICS region, `CICSTOR1`, and that test batch jobs trust the personal certificate of the z/OS Gateway daemon.

The import process is described in 10.4.2, “SSL application to Gateway daemon” on page 189 for the Gateway daemon on z/OS, and Java client applications, and again for the Gateway daemon on Multiplatforms in 11.5.4, “Secure the IPIC connection” on page 248.

9.4.1 Create a key ring for CICS

Use **`DFH$RING`** to create a key ring and a CA signer certificate for the CICS region `TOR1`. This signer certificate is a fundamental component of the overall configuration, because it also is required to be imported to the Gateway daemon key ring to establish a secure IPIC connection.

To build a key ring in the RACF database that is suitable for use in a CICS region, access is required to the appropriate profiles in the `FACILITY` class. For details, see the CICS TS for z/OS Information Center topic, *Setting up profiles in RACF*:

http://ibm.co/1yEmB5s

Make a working copy of the CICS TS sample REXX script, **SDFHSAMP(DFH$RING)**. On our test system, it is stored in the partitioned data set CICSSYSP.CICS.EXEC. Edit **DFH$RING**, making the changes shown in Example 9-11.

Example 9-11 Values used in DFH$RING to create an internal signer certificate

```
organization = 'Leading Edge CICS Enterprise'
department   = 'CICS Test Department'
certifier    = 'CICS-Sample-Certification'
city         = 'Gotham City'
state        = 'Xanadu'
country      = 'US'
```

To invoke the **DFH$RING** REXX script from TSO, use the following syntax:

```
EXEC 'hlq.SDFHSAMP(DFH$RING)' 'firstname lastname webservername [ FORUSER(userid)
] '
```

To create the key ring for the CICSTOR1 region user ID CICSRS3, issue the **EXEC** command:

```
EXEC 'CICSSYSP.CICS.EXEC(DFH$RING)' 'Cics Cicstor1 CICSTOR1 FORUSER(CICSRS3)'
```

The **DFH$RING** REXX script produces output detailing each of the four certificates that it has created and finally provides a summary of the new key ring, shown in Example 9-12.

Example 9-12 The new CA signer certificate and key ring for CICSTOR1

```
Digital ring information for user CICSRS3:

  Ring:
       >Cics.Cicstor1<
  Certificate Label Name              Cert Owner     USAGE      DEFAULT
  -------------------------------     ------------   --------   -------
  Verisign Class 1 Primary CA         CERTAUTH       CERTAUTH     NO
  Verisign Class 2 Primary CA         CERTAUTH       CERTAUTH     NO
  IBM World Registry CA               CERTAUTH       CERTAUTH     NO
  CICS-Sample-Certification           CERTAUTH       CERTAUTH     NO
  Cicstor1-Web-Server                 ID(CICSRS3)    PERSONAL     NO
  Cicstor1-IP-Connection              ID(CICSRS3)    PERSONAL     NO
  Cicstor1-Default-Certificate        ID(CICSRS3)    PERSONAL     YES
```

The CA signer certificate with label CICS-Sample-Certification has been generated, and then used to sign three personal certificates, for use by CICS region resources. In this book, we will use the Cicstor1-IP-Connection certificate to provide server authentication on a secure IPIC connection.

For more information about the CICS-supplied REXX script **DFH$RING**, see the topic, *Building a key ring with certificates using DFH$RING*, in the CICS Transaction Server for z/OS Information Center topic:

http://ibm.co/1yEqJTg

9.4.2 The KEYRING system initialization parameter

Before the CICS region can accept secure SSL connection requests, it must be configured to use those security resources created in 9.4.1, “Create a key ring for CICS” on page 163. The **KEYRING** system initialization parameter identifies a System Authorization Facility (SAF) certificate repository for use by secure TCPIPSERVICEs and IPCONN resources, installed within a CICS region.

We set the **KEYRING** system initialization parameter and we enabled security in our CICS region by using system initialization table (SIT) overrides:

```
KEYRING=Cics.Cicstor1
SEC=YES
```

The CICS log message, DFHXS1103, confirms that security is active in the CICS region:

```
DFHXS1103I S20351T1 Default security for userid CICSUSER has been established.
```

9.4.3 Configure a TCPIPSERVICE for SSL

This section highlights the minimum requirements to implement server authentication in CICS TS.

The TCPIPSERVICE definition in CICS TS includes the **SSl** parameter, with possible values of Yes, No, or Clientauth. Predefined or auto-installed IPIC connections are secure when the associated TCPIPSERVICE includes an **SSL** parameter of Yes or Clientauth.

> **Note:** Both the TCPIPSERVICE and the IPCONN resource types in CICS TS include an **SSl** attribute, with possible values of Yes, No, or Clientauth, and a **Ciphers** attribute. Only the **TCPIPSERVICE** attribute values are relevant to IPIC connections with CICS TG components (Gateway daemon or a local mode application).
>
> At run time, the IPCONN resource shows the **SSltype** attribute with a value Nossl, regardless of whether the IPIC connection to the CICS TG component is using SSL.

On the CICSTOR1 region, create a TCPIPSERVICE definition GW1IPICS in group T1IPIC by issuing the following command from a CICS terminal:

```
CEDA DEFINE TCPIPSERVICE(GW1IPICS) GROUP(T1IPIC)
```

We accepted default values for all parameters, except the following parameters:

- **POrtnumber**: 9001

  The port on which TOR is listening for inbound secure IPIC connection requests.

- **PROtocol**: IPic

  The protocol to be used by partner systems connecting through this service.

- **Backlog**: 00010

  Allow a sensible number of partner systems to queue for connection. Consider the number of independent IPIC connections likely to be established through this TCPIPSERVICE.

Example 9-13 on page 166 shows the SECURITY section of a TCPIPSERVICE definition, where the **SSl** attribute value is set to Yes and the **CIphers** attribute is the default set for STRONG encryption. The certificate with the Cicstor1-IP-Connection label is signed by the same CA signer certificate, CICS-Sample-Certification that is used to sign the Gateway daemon personal personal certificate in “Create and populate the Gateway daemon key ring” on page 186.

Example 9-13 Enabling SSL on an IPIC TCPIPSERVICE definition

```
SECURITY
  SSl            : Yes                Yes | No | Clientauth
  CErtificate    : Cicstor1-IP-Connection
  (Mixed Case)
  PRIvacy        : Supported          Notsupported | Required | Supported
  CIphers        : 050435363738392F303132330A1613100D0915120F0C0306
  AUthenticate   :                    No | Basic | Certificate | AUTORegister
                                      | AUTOMatic
  Realm          :
 (Mixed Case)
 ATtachsec      :                    Local | Verify
```

The TCPIPSERVICE definition was then installed by issuing the following command from a CICS terminal:

```
CEDA INSTALL TCPIPSERVICE(GW1IPICS) GROUP(T1IPIC)
```

The CICS log messages shown in Example 9-14 indicate that the TCPIPSERVICE resource was successfully installed and opened.

Example 9-14 CICSTOR1 region log messages for IPIC connection GW1IPICS

```
DFHSO0133 10/29/2013 14:12:25 CICSTOR1 TCPIPSERVICE GW1IPICS has been installed.
DFHSO0107 10/29/2013 14:12:25 CICSTOR1 TCPIPSERVICE GW1IPICS has been opened on
   port 09001 at IP address 192.168.0.1.
```

Then, verify the status of the TCPIPSERVICE resources by using the following command from a CICS terminal:

```
CEMT INQUIRE TCPIPSERVICE(GW1*)
```

The output is shown in Figure 9-4, where the TCPIPSERVICE without security, GW1IPIC, has an **Ssltype** attribute with a value of Nos, which is a contraction of Nossl in this summary view. The TCPIPSERVICE with security, GW1IPICS, shows an **Ssltype** attribute with a value of Ssl.

```
I TCPIPSERVICE
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Tcpips(GW1IPIC ) Ope Por(09000) Ipic Nos Tra(CISS)
    Con(00004) Bac( 00010 ) Maxd( 000000 ) Urm( NO       )
 Tcpips(GW1IPICS) Ope Por(09001) Ipic Ssl Tra(CISS)
    Con(00000) Bac( 00010 ) Maxd( 000000 ) Urm( NO       ) Sup
```

Figure 9-4 Verifying the SSL attribute of the installed TCP/IP services

9.5 Configuring IPIC autoinstall for WebSphere Application Server clusters

The configuration described in Chapter 13, “High availability configuration” on page 295 uses a WebSphere Application Server cluster in remote mode. It also includes additional considerations when using a WebSphere Application Server cluster in local mode to establish IPIC connections to CICS TS, where a single connection factory definition is installed to each member of the cluster.

This local mode cluster configuration encounters issues when the connection factory definition, following the preferred practice, defines an APPLID. Problems arise using IPIC autoinstall with the CICS default IPIC autoinstall user program `DFHISAIP`, or using predefined IPIC connections, due to a naming clash when an attempt is made to establish a second IPCONN from a partner with the same APPLID. For a more detailed explanation, see 4.4, “IPIC autoinstall for WebSphere Application Server clusters” on page 64.

For the purpose of implementing 13.6, “WebSphere Application Server cluster” on page 314, the CICS regions are configured for IPIC autoinstall, using the new IPIC autoinstall sample program, `DFHOISAI`, introduced in CICS TS for z/OS V5.1.

We were able to use the sample as provided without requiring further customization. The only limitation is that the value of the **`Applid`** attribute, specified in the custom properties of the Java EE Connector Architecture connection factory definition, must be no more that seven characters. This limit is because the sample autoinstall program appends a unique integer suffix to the **`Applid`** value specified in the request and the IPCONN resource definition name is restricted to eight characters.

The source code for the CICS TS sample autoinstall user program, which is named `DFHOISAI`, to support predefined connection templates is provided in the CICS PDS `CICSHLQ.SDFHSAMP` and the pre-compiled load module is provided in `CICSHLQ.SDFHLOAD`.

The WebSphere Application Server cluster configuration described in Chapter 13 uses a Java EE Connector Architecture connection factory with an **`Applid Qualifier`** value of `CTGWASZ` and an **`Applid value`** of `ITSOZOS`. To avoid duplicate IPCONN resources being installed, the CICS TS-supplied IPIC autoinstall program `DFHOISAI` is required.

The following steps describe how our CICS region was configured to use the `DFHOISAI`:

> **Note:** The `PROGRAM` definition and installation steps can be skipped if program autoinstall is enabled.

1. `DFHOISAI` is supplied precompiled in the CICS TS load library, `CICSHLQ.SDFHLOAD`, and is already available without modifying `DFHRPL`.
2. Create a PROGRAM resource definition for `DFHOISAI` in the same way that you create a resource definition in “PROGRAM definitions for the samples” on page 158:
   a. Issue the following command to define program `DFHOISAI` in group `T1IPIC`:

      `CEDA DEF PROGRAM(DFHOISAI) GROUP(T1IPIC)`

   b. Accept the default values for all parameters, except for the following parameters:
      - **`Language`**: `COBOL`
      - **`DAtalocation`**: `Any`

3. Install the PROGRAM resource definition by issuing the following command:

   CEDA INSTALL PROGRAM(DFHOISAI) GROUP(T1IPIC)

4. Create a TCPIPSERVICE resource definition, in the same way as described in 9.2.3, “TCPIPSERVICE resource definition” on page 151:

   a. Issue the following command to define the TCPIPSERVICE resource `WASIPIC` in group `T1IPIC`:

      CEDA DEF TCPIPSERVICE(WASIPIC) GROUP(T1IPIC)

   b. Accept the default values for all parameters, except for the following parameters:

      - **Urm**: `DFHOISAI`

        The CICS TS sample autoinstall user program to support predefined connection templates.

      - **POrtnumber**: `9002`

        The port on which TOR is listening for inbound IPIC requests from WebSphere Application Server.

      - **PROtocol**: `IPic`

        The protocol to be used by partner systems that connect through this service.

      - **Backlog**: `00010`

        Allow a sensible number of partner systems to queue for connection. A backlog value of at least `4` (two sockets for each IPIC connection) will suffice for our pair of WebSphere Application Server servant regions in 13.6, “WebSphere Application Server cluster” on page 314, but 10 gives you space for growth.

      - **Host**: `lp01.redbooks.ibm.com`

        Bind the listening port to a static IP address for LPAR `lp01` to avoid binding to multiple addresses. For further details, see “Binding of listener ports” on page 300.

5. Install the TCPIPSERVICE resource definition by issuing the following command:

   CEDA INSTALL TCPIPSERVICE(WASIPIC) GROUP(T1IPIC)

6. Create an IPCONN template resource definition, in the same way as described in 9.2.4, “IPCONN resource definition” on page 153. The IPCONN template name must match the Applid Qualifier value defined in the Java EE Connector Architecture connection factory custom properties definition in WebSphere Application Server, in this example, `CTGWASZ`:

   a. Issue the following command to define the IPCONN resource `CTGWASZ` in group `T1IPIC`:

      CEDA DEF IPCONN(CTGWASZ) GROUP(T1IPIC)

   b. Accept the default values for all parameters, except for the following parameters:

      - **Ipconn**: `CTGWASZ`

        Match the Applid Qualifier value specified in the Java EE Connector Architecture connection factory custom properties definition in WebSphere Application Server.

      - **APplid**: `ITSOZOS`

        Match the Applid value specified in the Java EE Connector Architecture connection factory custom properties definition in WebSphere Application Server. This must be a maximum of seven characters to allow the autoinstall program to append a unique integer suffix.

      - **Tcpipservice**: `WASIPIC`

        The name of the TCPIPSERVICE resource definition defined previously.

- **Receivecount**: 100

  Support up to 100 concurrent application requests over this connection.

7. Install the IPCONN resource definition by issuing the following command:

   ```
   CEDA INSTALL IPCONN(CTGWASZ) GROUP(T1IPIC)
   ```

The WebSphere Application Server cluster configuration described in Chapter 13, “High availability configuration” on page 295 uses a Java EE Connector Architecture connection factory with an `Applid Qualifier` value of `CTGWASZ` and an `Applid` value of `ITSOZOS`.

10

CICS Transaction Gateway for z/OS configuration

This chapter describes a basic remote mode configuration of CICS Transaction Gateway for z/OS (CICS TG), using an IPIC connection to CICS Transaction Server for z/OS (CICS TS) and corresponding verification steps. It then describes two alternative security configurations before adding support for two-phase commit (XA) transactions.

This chapter covers the following topics:

10.1 Configure the Gateway daemon

This section explains how to configure a Gateway daemon for z/OS with an non-secure IPIC connection to a CICS TS for z/OS terminal-owning region (TOR). It introduces the common method for starting the Gateway daemon, the **CTGBATCH** utility. It is assumed that the CICS region, CICSTOR1, is configured as described in Chapter 9, “Configuring IPIC in CICS Transaction Gateway for z/OS” on page 147, including an IPIC connection for the Gateway daemon to use.

You can configure the Gateway daemon on z/OS through a combination of the following resources:

- Configuration file
- Environment variables
- Java properties
- Overrides
- Started procedure or job control language (JCL)
- System defaults
- Language Environment runtime options

The Gateway daemon determines its operational parameters from the configuration file, overrides, and Java properties. The environment in which the Gateway daemon runs is determined by a combination of other z/OS-specific factors.

The various files that encapsulate a Gateway daemon configuration can be stored in various locations, including both the UNIX file system and MVS data sets. The examples in this book place the configuration file and environment variables together in a single MVS-partitioned data set (PDS), because this placement simplifies the task of configuration and allows backup or version control of the configuration data.

10.1.1 Configuration summary

The workload requires 200 concurrently connected clients, with the capacity to process up to 100 concurrent requests, and a maximum 64 KB payload size. To support payloads greater than 32 KB, IPIC connectivity is required. Table 10-1 summarizes the basic configuration details that are required for the Gateway daemon.

Table 10-1 Gateway daemon basic configuration values

Attribute	Value
APPLID qualifier	CTGALP00
APPLID	ITSOCTG0
TCP handler port number/bind address	2016/lp01.redbooks.ibm.com
SSL handler port number (bind address)	8070/lp01.redbooks.ibm.com
Stats handler port number (bind address)	2980/lp01.redbooks.ibm.com
Maximum connection manager threads	200
Maximum worker threads	100
IPICSERVER host/port/APPLID (non-SSL)	lp01.redbooks.ibm.com/9000/S2035T1
IPICSERVER host/port/APPLID (SSL)	lp01.redbooks.ibm.com/9001/S2035T1

Product system files

CICS TG for z/OS is SMP/E maintained and consists of several MVS data sets and a product directory on the z/OS UNIX file system. Table 10-2 lists the location of the CICS TG installation resources.

Table 10-2 z/OS product files

Product/system files	Location
MVS	`CTG.V9ROMO`
z/OS UNIX file system	`/usr/lpp/cicstg/ctg900`
z/OS UNIX file system	`/usr/lpp/java/J7.0/bin`

CICS TG configuration resources

CICS TG for z/OS configuration resources can be stored in MVS data sets or on the z/OS UNIX file system. Opting for a PDS allows the CICS TG configuration file and the environment variables to be stored in one location, which provides easier cloning, version control, and backup. The procedure JCL must be in a standard location.

Table 10-3 lists the locations selected for this implementation.

Table 10-3 CICS TG configuration resources

File	Location	Description
CTG0CONF	`CICSRS4.CTG900.CONFIG`	CICS TG configuration file
CTG0ENV	`CICSRS4.CTG900.CONFIG`	CICS TG environment variables
CTGGWX	`SYS1.PROCLIB`	Procedure JCL to start the Gateway daemon

10.1.2 The CTGBATCH launcher

On z/OS, the Gateway daemon runs as a UNIX System Services process and can be started using a variety of techniques. Given a UNIX System Services runtime environment and the required environment variables, invoking the **`ctgstart`** shell script attempts to start the Gateway daemon. However, the preferred method is to use the CICS TG-supplied MVS utility, **`CTGBATCH`**, which is in the `SCTGLOAD` data set.

The **`CTGBATCH`** utility provides a z/OS runtime “container” for the Gateway daemon, allowing integration with fundamental z/OS facilities, such as the z/OS console, and logging to Job Entry Subsystem (JES) output. It establishes the runtime environment for the Gateway daemon, controlling the specific level of Java, logging destinations, and overall memory resources.

The **`CTGBATCH`** utility takes a UNIX System Services path to the target executable (for example, **`ctgstart`**) through the **`PARM`** parameter. It sets environment variables that are supplied through the `STDENV DD` card and then executes the target script or program.

Example 10-1 shows a **`CTGBATCH`** job step for executing the **`ctgstart`** shell script.

Example 10-1 The CTGBATCH job step

```
// SET CTGHLQ='CTG.V9ROMO'
// SET CTGHOME='/usr/lpp/cicstg/ctg900'
// SET CTGENV='CICSRS4.CTG900.CONFIG(CTG0INI)'
/ SET LEOPTS='/'
```

```
//CTGRUN   EXEC PGM=CTGBATCH,REGION=500M,
//         PARM='&LEOPTS.&CTGHOME./bin/ctgstart'
//STEPLIB  DD DSN=&CTGHLQ..SCTGLOAD,DISP=SHR
/STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDENV   DD    DSN=&CTGENV.,DISP=SHR
```

The **CTGBATCH** utility is functionally similar to the **BPXBATCH** z/OS-supplied utility and it can be used to run any UNIX System Services shell script or executable program. However, it has special features that are designed for use with the Gateway daemon. These features include a debug option that is useful for checking environment variables and runtime settings. In addition to the default English language support, the **CTGBATCH** utility provides globalization for Chinese and Japanese.

Another key difference is the structure of the **PARM** string. The **CTGBATCH** parameter string is split into the following parts:

- Language Environment runtime options
- The target z/OS UNIX program or script
- Parameters for the target program or script

These first two components of the PARM string are delimited by a forward-slash character (/). Using this character can be confusing, because the second component, the target program, or script can be a fully qualified z/OS UNIX path that starts with a forward-slash character. It is convenient to include the Language Environment option string delimiter in a variable, as shown in Example 10-1 on page 173, where an empty set of Language Environment options together with the delimiter is represented as a forward-slash character.

The Gateway daemon writes informational log messages to a UNIX standard out destination. Warning and error logs messages are written to the UNIX standard error destination. You can map these standard output destinations to a JES spool output, hierarchical file system (HFS) files, or MVS data sets using **CTGBATCH DD** cards `STDOUT` and `STDERR`. The examples in this book map these outputs to JES spool output using `STDOUT` and `STDERR DD` statements, as shown in Example 10-1 on page 173.

10.1.3 Configuration file

The CICS TG configuration file uses a common format, syntax, and set of resources for each of the CICS TG products. There are product-specific components and limitations, but this topic describes a configuration based on IPIC connections.

Note: Although this configuration file is for use with the CICS TG for z/OS product, it is also usable with the Multiplatform Gateway daemon simply by adding definitions for log files.

When storing the configuration in an MVS data set, be careful with the maximum line length due to the file `LRECL` value. The `SCTGSAMP(CTGCONF)` sample shows how to use continuation characters for those entries that typically require multiple lines.

PRODUCT section

In Example 10-2, the `PRODUCT` section includes configuration details that apply to all components of the Gateway daemon at run time. The `APPLID` qualifier and `APPLID` are mandatory only when XA support is enabled, but this setting is a preferred practice when using IPIC connections to easily identify the Gateway daemon as an IPIC partner.

Example 10-2 The CICS TG configuration file PRODUCT section

```
SECTION PRODUCT
    Applid = ITSOCTGO # The APPLID of the Gateway daemon
    ApplidQualifier = CTGALPOO  # The APPLID qualifier of the Gateway daemon
    DefaultServer = CICSTOR1    # The default server used by the Gateway daemon
ENDSECTION
```

GATEWAY section

In Example 10-3, the `GATEWAY` section encapsulates important characteristics, such as the connection manager and work thread pools, and the protocol handlers for the ports of entry.

Example 10-3 The CICS TG configuration file PRODUCT section

```
SECTION GATEWAY
# Define Gateway daemon thread pools
    InitConnect = 1
    MaxConnect = 200
    InitWorker = 1
    MaxWorker = 100

# Protocol handler for TCP/IP requests
    protocol@tcp.handler = com.ibm.ctg.server.TCPHandler
    protocol@tcp.parameters = bind=lp01.redbooks.ibm.com;\
                              connecttimeout=2000;\
                              idletimeout=600000;\
                              pingfrequency=60000;\
                              port=2016;\
                              solinger=10;

# Protocol handler for statistics API requests
    protocol@statsapi.handler = com.ibm.ctg.server.RestrictedTCPHandler
    protocol@statsapi.parameters = bind=lp01.redbooks.ibm.com;\
                                   connecttimeout=2000;\
                                   maxconn=5;\
                                   port=2980;

# Disable the reading of input from UNIX System Services stdin
    NoInput = on
ENDSECTION
```

There is no need to configure a logging destination for the Gateway daemon on z/OS in the configuration file, because the standard `stdout` and `stderr` are used automatically. Informational log messages are written to `stdout` and are mapped by the **`CTGBATCH`** `STDOUT DD` statement. Similarly, warning and error log messages are written to `stderr` and mapped by the **`CTGBATCH`** `STDERR DD` statement.

The bind attribute on the protocol handler definitions accepts either domain name server (DNS) host names or IP addresses. Using a host name is more readable and flexible, allowing actual IP addresses to be changed without updating such references. However, if the host name cannot be resolved, the Gateway daemon will fail to initialize, issuing the error message:

```
CTG6525E Unable to start handler for the tcp: protocol, port: 0, because:
[java.lang.IllegalArgumentException: bind= unknown.itso.ibm.com]
```

IPIC connection

Each IPIC connection to a CICS server is defined by an `IPICSERVER` section (Example 10-4) of the configuration file.

Example 10-4 An IPIC server definition

```
SECTION IPICSERVER = CICSTOR1
   Description = IPIC Server
   Hostname = lp01.redbooks.ibm.com
   Port = 9000
   TcpKeepAlive = Y
   Ssl = N
   CICSApplid = S20351T1
   CICSApplidQualifier = USIBMSC
   SendSessions = 50
ENDSECTION
```

10.1.4 Environment variables

The **`CTGBATCH`** utility reads environment variables from the `STDENV DD` card that can be mapped to either an MVS data set, UNIX file system, or in line with the JCL. Multi-line values can be specified using the back slash (\) special continuation character.

The Gateway daemon must run under a user ID with an OMVS segment.

> **Note:** The **`CTGBATCH`** utility does not automatically execute either the default profile (`/etc/profile`) or the user's own profile (`.profile`) before executing the target script or program. Therefore, the target script or UNIX System Services program does not start with the same environmental settings observed when that user ID starts an OMVS session. For this reason, the **`CTGBATCH`** utility must be configured with the most fundamental PATH settings, such as `PATH=/bin:/usr/bin`, to successfully execute a shell script, such as **`ctgstart`**.

Numerous environment variables are available. However, for this book, examples run with external CICS interface (EXCI) disabled. Therefore, a minimal set of values are required, as shown in Example 10-5.

Example 10-5 CICS TG environment variables

```
PATH=/bin:/usr/lpp/java/J7.0/bin
CICSCLI=//'CICSRS4.CTG900.CONFIG(CTG0CONF)'
CTG_EXCI_INIT=NO
CLASSPATH=/usr/lpp/cicstg/ctg90/classes/ctgsamples.jar
COLUMNS=80
TMPDIR=/tmp
TZ=EST0EDT
```

```
_BPX_SHAREAS=YES
_BPXK_SETIBMOPT_TRANSPORT=TCPIP
_CEE_DMPTARG=/tmp/jvmdump
```

The details of the environment variables are listed:

PATH
Must be configured to allow target executable shell scripts to find the fundamental UNIX commands, such as **echo** or **awk**, which are in the `/bin` directory. The **ctgstart** script also requires the **Java** command, from the appropriate version of the SDK.

Value for PATH environment variable: The value specified for the **PATH** environment variable is explicit. The **CTGBATCH** utility does not append the specified value with any system default value.

CICSCLI
Specifies the location of the CICS TG configuration file that defines the core runtime characteristics of the Gateway daemon.

CTG_EXCI_INIT
Controls whether the EXCI protocol is to be available. Setting this environment variable to `YES` requires multiple, further environment variables to be considered and set. It also limits the maximum number of worker threads that are available to the Gateway daemon to the value of `LOGONLIM` (100 is the default value. CICS imposes a maximum of 250.) The examples in this book implement a solution based on IPIC connectivity and do not require EXCI.

CLASSPATH
Optional: Used to specify the location of Java code for use with Gateway daemon user exits, such as the security exits, request monitor exits, or CICS request exit. We added the `ctgsamples.jar` file to use the precompiled CICS TG sample, `BasicCICSRequestExit`.

Product files and the CLASSPATH variable: You do not need to specify the location of the CICS TG product files (`ctgserver.jar` and `ctgclient.jar`). These files are appended automatically to the *classpath* variable by the **ctgstart** script.

TMPDIR
Defines a location on the file system for the z/OS UNIX file system. This location must always have free space that is writable by the Gateway daemon user ID.

When TMPDIR becomes full: If the z/OS UNIX file system location used by `TMPDIR` becomes full, fundamental shell script statements cannot be evaluated. In this case, the Gateway daemon might fail to start, which can be confusing if `TMPDIR` happens to be full intermittently, because the Gateway daemon might fail to initialize. Then, the Gateway daemon subsequently succeeds without a change to the overall configuration.

_BPX_SHAREAS	Controls how z/OS UNIX processes are mapped to MVS address spaces. Setting this option to `YES` ensures that the Gateway daemon processes run within a single MVS address space. This setting is more efficient and makes it clear which job name to use with the **MODIFY** command for system administration tasks.
_BPXK_SETIBMOPT_TRANSPORT	Specifies the job name of the TCPIP address space for z/OS UNIX programs to use by default. This setting is useful when the Gateway daemon is running in a logical partition (LPAR) with multiple TCP/IP address spaces.
COLUMNS	Is used only by the **ctgstart** script to format the maximum width of any log messages issued by the script itself. The default is 40, but a setting of 80 is more appropriate for running under the **CTGBATCH** utility.
TZ	Determines the time zone that is used in time stamps for log messages and diagnostic traces that are produced by the Gateway daemon. For eastern standard time, use `EST0EDT`.
_CEE_DMPTARG	Specifies the z/OS UNIX path for Java Core dump, Java Heap dump, and System dumps generated automatically or by the CICS TG dump commands. Similar to `TMPDIR`, this setting needs to be a location on the z/OS UNIX file system that is writable by the Gateway daemon user ID and with plenty of free space.

For more information about the available environment variables and their use, see the CICS TG for z/OS Information Center:

http://pic.dhe.ibm.com/infocenter/cicstgmp/v9r0/index.jsp

10.1.5 Overriding the configuration file

Certain configuration file values can be overridden at run time by passing parameters to the **ctgstart** script. Unfortunately, the `JCL PARM` attribute of the **PGM** command used to pass parameters to the script has a maximum length of 100 characters. After specifying the fully qualified path to the **ctgstart** script, little space remains for useful configuration override values. The **CTGSTART_OPTS** environment variable provides a solution for this limitation, allowing full use of the facility to override CICS TG configuration file settings.

This environment variable allows you to use a CICS TG configuration file as a template for multiple Gateway daemons within a Gateway group. Each Gateway daemon instance within the group shares a common set of characteristics. Example 10-6 shows how to use the **CTGSTART_OPTS** environment variable to specify two **ctgstart** override options, split over two lines in the `STDENV` data.

Example 10-6 Overriding the CICS TG configuration file

```
CTGSTART_OPTS=-j-Xmx256M \
-requestExits=com.ibm.ctg.samples.requestexit.BasicExitSample
```

10.1.6 Started procedure and Gateway daemon user ID

For test or evaluation purposes, starting a Gateway daemon using a JOB can be acceptable. However, to avoid affinities to specific user IDs, it is a preferred practice to use a started task. This method allows a Gateway daemon to be used by any administrator with the required authority, and the Gateway daemon always runs under a dedicated started task user ID.

It is also a preferred practice to define a security profile that encapsulates the set of permissions that are required to run a Gateway daemon and to connect the started task user ID to that security profile.

In RACF terms, create a group, `CTGGRP`. In the `FACILITY` class, permit the group with `READ` access to the `BPX.STOR.SWAP` profile and `UPDATE` access to the `CTG.RRMS.SERVICE` profile, using the commands shown in Example 10-7. This example creates a started task `CTGGW` user ID and connects it to the `CTGGRP` group.

Example 10-7 PERMIT commands profiles in the FACILITY class

```
PERMIT BPX.STOR.SWAP CLASS(FACILITY) CTGGRP ACCESS(READ)
PERMIT CTG.RRMS.SERVICE CLASS(FACILITY) CTGGRP ACCESS(UPDATE)
```

You can use the `BPX.STOR.SWAP` profile to restrict those address spaces that can make themselves non-swappable. Use this method for the Gateway daemon; otherwise, the Gateway daemon can become unresponsive to application requests following a period of inactivity.

The `CTG.RRMS.SERVICE` profile can restrict access to authorized recoverable resource management services (RRMS) capabilities. These capabilities are required by Gateway daemons that are enabled with XA transaction support.

Maintenance note: When applying maintenance to `SCTGLINK(CTGINIT)`, a separate user ID with CONTROL access to `CTG.RRMS.SERVICE` is required to restart `CTGRRMS`.

You can use the **`TSO RLIST FACILITY`** command with the sub-option `AUTHUSER` to list the user IDs or groups and their access to a particular profile. Example 10-8 shows the output from such a command relevant to the `CTGGRP` group.

Example 10-8 Extract from the output of TSO RLIST FACILITY BPX.STOR.SWAP AUTHUSER

```
CLASS      NAME
-----      ----
FACILITY   BPX.STOR.SWAP

USER      ACCESS   ACCESS COUNT
----      ------   ------ -----
CTGGRP    READ        000000
```

You can use a similar command to verify access to the `CTG.RRMS.SERVICE` profile. The started task JCL uses the **`CTGBATCH`** launcher program to call the **`ctgstart`** shell script and provides access to all of the configuration resources that are required by the Gateway daemon.

Example 10-9 on page 180 shows a sample that is stored as the `CTGGWX` member of the `SYS1.PROCLIB` data set.

Example 10-9 Gateway daemon started task JCL

```
//CTGGWX  PROC CTGID='0'
//* Gateway daemon resources
// SET CTGHLQ='CTG.V9ROMO'
// SET CTGHOME='/usr/lpp/cicstg/ctg900'
// SET CTGENV='CICSRS4.CTG900.CONFIG'
//* Use default LE run-time options
// SET LEOPTS='/'
//* Call CTGBATCH
//CTGRUN   EXEC PGM=CTGBATCH,REGION=500M,
//         PARM='&LEOPTS.&CTGHOME./bin/ctgstart'
//STEPLIB  DD DSN=&CTGHLQ..SCTGLOAD,DISP=SHR
//* Map stdout and stderr to JES  output
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//* Load CTGID-specific environment variables
//STDENV   DD DSN=&CTGENV.(CTG&CTGID.ENV),DISP=SHR
//* Uncomment for CTGBATCH debug output
//*CTGDBG   DD DUMMY *
// PEND
//
```

A started task can be associated with a specific user ID in numerous ways. With RACF, you can use the `STARTED` general resource class to associate a specific user ID with started tasks by name. The Gateway daemon `CTGGW` started task user ID is associated with any started task beginning with *CTGGW*, using commands shown in Example 10-10.

Example 10-10 Associating a user ID with a started task

```
SETR GENERIC(STARTED)
RDEFINE STARTED CTGGW*.** STDATA(USER(CTGGW))
SETROPTS RACLIST(STARTED) REFRESH
```

The Gateway daemon is started using the **`/S CTGGW`** MVS start command. By default, the `CTGID` JCL variable is set to 0 (zero). Therefore, the environment variables are taken from the `CTG0ENV` member of the `CICSRS4.CTG900.CONFIG` data set. The **`CICSCLI`** environment variable defines the configuration file as the `CTG0INI` member of the `CICSRS4.CTG900.CONFIG` data set.

Example 10-11 shows key log messages from the Gateway daemon initialization, reflecting the configuration resources in use.

Example 10-11 Identifying configuration resources from the Gateway daemon log

```
IEFC653I SUBSTITUTION JCL - DSN=CICSRS4.CTG900.CONFIG(CTG0ENV),DISP=SHR

CTG8400I Using configuration file //'CICSRS4.CTG900.CONFIG(CTG0INI)'

CTG6577I Java details: Version=1.7.0 - 31-bit, Path=/usr/lpp/java/J7.0/bin/java
```

The `CTGID` JCL variable allows the same JCL started task to start multiple Gateway daemon instances. Using the standard MVS start command, this method leads to multiple Gateway daemon address spaces using the same `CTGGWX` job name. This same job name causes a problem for system administration, because only one address space receives a modify command directed to a specific job name.

To circumvent this problem, two possible options allow manipulation of the JOB name that identifies the started task, through options on the **START (/S)** command.

- Give each instance a unique **JOBNAME** parameter on the **START** command. Then, use the job name on subsequent modify commands. For example, specify the CTGGWO job name:

  ```
  /S CTGGWX,JOBNAME=CTGGWO
  /F CTGGWO,APPL=SHUT
  ```

- Qualify the MVS start and modify commands with an identifier, and use the identifier on subsequent modify commands. For example, specify the GWO identifier:

  ```
  /S CTGGWX.GWO
  /F CTGGWX.GWO,APPL=SHUT
  ```

10.2 Testing the Gateway daemon

Now, you can test the Gateway daemon that you configured in 10.1, “Configure the Gateway daemon” on page 172 using the CTGTESTR sample JOB. Copy the sample JOB from the CTG.V9ROMO.SCTGSAMP product installation data set to a working data set. You must customize the CTGTESTR sample to specify Gateway daemon-specific parameter values. It is a preferred practice to edit a copy of the SMP/E-maintained version.

10.2.1 Start the Gateway daemon

Verify that the Gateway daemon was initialized successfully. Verify that the Gateway daemon is listening for application requests on the intended port number. Example 10-12 shows those key log messages from the Gateway daemon initialization, indicating these details.

Example 10-12 Verifying key Gateway daemon configuration and successful initialization

```
CTG8426I The APPLID is ITSOCTGO
CTG8427I The APPLID qualifier is CTGALPOO
CTG8667I EXCI Support is disabled
CTG6981I Successfully initialized JNI library
CTG6502I Initial Connection managers = 1, Maximum Connection managers = 200
CTG6526I Initial workers = 1, Maximum workers = 100
CTG6505I Successfully created the initial connection manager and Worker threads
CTG6524I Successfully started handler for the tcp: protocol on port 2016
CTG6524I Successfully started handler for the statsapi: protocol on port 2980
CTG6512I CICS Transaction Gateway initialization complete
```

10.2.2 Test using the CTGTESTR job

The **CTGTESTR** sample job uses the **CTGBATCH** utility to invoke the **ctgtest** z/OS UNIX shell script. This shell script runs the EciB2 Java sample program (com.ibm.ctg.samples.eci.EciB2), with parameter values that specify a target Gateway daemon, CICS server, security credentials, CICS program, and data length.

The following inline STDENV data for the **CTGBATCH** utility configures CTGTESTR to send a request to CICSTOR1 through the Gateway daemon. As with the Gateway daemon started task JCL, the **CTGBATCH** PARM string is limited to 100 characters. CTGTESTR circumvents this limit by allowing parameters for the **ctgtest** shell script to be passed using the **CTGTEST_OPTS** environment variable.

Example 10-13 shows the name/value-pair parameters that are required by the `EciB2` sample program to drive the CICS program `EC01` in CICS server `CICSTOR1`.

Example 10-13 Customizing CTGTESTR through the environment variables

```
//STDENV   DD    *
PATH=/bin:/usr/lpp/java/J7.0/bin
CTGTEST_OPTS=jgate=tcp://localhost \
jgateport=2016 server=CICSTOR1 \
prog0=EC01 commarealength=18
/*
```

The output from CTGTESTR is written to the JES output, `STDOUT`. Example 10-14 shows successful test of the Gateway daemon that is configured in 10.1, “Configure the Gateway daemon” on page 172.

Example 10-14 CTGTESTR stdout output for CICS program EC01 without security

```
ctgtest: Executing EciB2 with parameters:
jgate=tcp://localhost jgateport=2016 server=CICSTOR1 prog0=EC01 commarealength=18

CICS Transaction Gateway Basic ECI Sample 2
 Test Parameters
CICS TG address : tcp://localhost:2016
Client security : null
Server security : null
CICS Server     : CICSTOR1
UserId          : null
Password        : null
Data Conversion : ASCII
COMMAREA        : null
COMMAREA length : 18
Number of programs given : 1
  [0] : EC01
Connect to Gateway
Successfully created JavaGateway
Call Programs
About to call : EC01
  COMMAREA    :
  extend_mode : 0
  LUW_token   : 0
  Commarea    : 20/06/13 10:09:24
Return code   : ECI_NO_ERROR(0)
Abend code    : null
Successfully closed JavaGateway
```

Because you can use `EciB2` to drive a user-specified CICS program, you must specify an adequate `COMMAREA` length. Undersized **`COMMAREA`** length parameters can cause the target CICS program to ABEND or produce unpredictable results.

Determining a COMMAREA length: In this case, the `EC01` CICS program (source code is in the CICS TG product z/OS UNIX `samples/server/ec01.ccp` file), it is easy to determine that a `COMMAREA` length of 18 bytes is adequate by inspecting the `DFHCOMMAREA` definition:

```
01  DFHCOMMAREA.
          05  LK-DATE-OUT      PIC X(8).
          05  LK-SPACE-OUT     PIC X(1).
          05  LK-TIME-OUT      PIC X(8).
          05  LK-LOWVAL-OUT    PIC X(1).
```

`EC01` includes defensive code to check the length of the input `COMMAREA`. Running **`CTGTESTR`** with a `COMMAREA` length of `10` bytes causes the CICS program `EC01` to fail. The `EciB2` sample returns the following details:

```
ECI_ERR_TRANSACTION_ABEND(-7)
Abend code    : ECOM
```

Important: A CICS program might not include defensive logic that is related to the `COMMAREA` input length. Supplying an insufficient length might lead to unpredictable behavior.

`EciB2` is a useful test tool because it is flexible and allows you to set a useful range of ECI request parameters. You can use it to invoke multiple CICS program calls in a sequence of independent synchronized on return calls or in a transactional manner by using an extended logical unit of work.

Example 10-15 shows the full `EciB2` usage information that is output when no parameters are supplied.

Example 10-15 EciB2 usage information output

```
ctgtest: Executing EciB2 with parameters:
CICS Transaction Gateway Basic ECI Sample 2
This sample can be used to test ECI calls to CICS applications from Java
Applications.
You can specify the Gateway URL and relevant ECI call parameters as input
to the application, and either call a single CICS program or call
multiple CICS programs within one extended Logical Unit of Work. The code
page of the COMMAREA flowed on the ECI call can be controlled as an input
parameter.
Note: jgateport is ignored for local mode (jgate=local:)
IPIC_url syntax is <protocol>://<host>:<port>, where protocol is "tcp"
or "ssl" and <host>:<port> represent the CICS IPIC TCPIP service definition
Usage:
  java com.ibm.ctg.samples.eci.EciB2  [jgate=gateway_URL]
                                      [jgateport=gateway_port]
                                      [clientsecurity=client_security_class]
                                      [serversecurity=server_security_class]
                                      [server=cics_server_name or IPIC_url]
                                      [userid=cics_user ID]
                                      [password=cics_password]
                                      [prog<0..9>=prog_name]
                                      [COMMAREA=comm_area]
                                      [COMMAREAlength=comm_area_length]
                                      [status]
                                      [trace]
```

```
                                    [ascii | ebcdic | asis]
Example:
  java com.ibm.ctg.samples.eci.EciB2 jgate=tcp://server.ibm.com jgateport=2016
   server=mycics prog0=EC01 COMMAREA=mydata userid=myuid password=mypwd
```

10.3 Configure basic security

The only change required to enable the verification of user ID and password (or password phrase) credentials is in the **USERAUTH** parameter of the CICS TS for z/OS Ipconn definition. Implementing security based on a prevalidated user ID without a password, using USERAUTH(IDENTIFY), is described in “Asserted identity using the secure CICS connection” on page 195.

10.3.1 CICS TS configuration for USERAUTH(VERIFY)

Setting USERAUTH to VERIFY causes the target CICS region to perform authentication on the security credentials included in the ECI request. The change is minimal, because the configuration described in this chapter uses IPIC connections exclusively.

When EXCI is enabled: When EXCI is enabled, ECI requests destined for a CICS server through an EXCI connection are authenticated by the Gateway daemon rather than the target CICS region. Additional configuration of the Gateway daemon is required.

Example 10-16 shows the SECURITY section of the IPConn definition, which represents the IPIC connection from the Gateway daemon to CICSTOR1. During the initial configuration, without security, the **Userauth** parameter value was Local. It is now changed to Verify.

Example 10-16 Configuring the Ipconn to perform password verification

```
OVERTYPE TO MODIFY                                        CICS RELEASE = 0680
  CEDA  ALter Ipconn( CTG0 )
+  INservice    ==> Yes                Yes | No
  SECURITY
   SSl          ==> No                 No | Yes
   CErtificate  ==>                                                (Mixed Case)
   CIphers      ==>
   Linkauth     ==> Secuser            Secuser | Certuser
   SECurityname ==>
   Userauth     ==> Verify             Local | Identify | Verify | Defaultuser
   IDprop       ==> Notallowed         Notallowed | Optional | Required
  RECOVERY
   Xlnaction    ==> Keep               Keep | Force
  MIRROR TASK PROPERTIES
   MIrrorlife   ==> Request            Request | Task | Uow
```

Before this definition can be updated, the IPConn must be set out of service and discarded. Before the Ipconn can be set out of service, it must be in a released state. To prevent the Gateway daemon from reconnecting automatically, set the TCPIPService that is associated with the Ipconn definition to be closed using the **CEMT** command:

```
CEMT SET TCPIPS(GW1IPIC) CLOSED
CEMT SET IPCONN(CTG0) OUT
```

The Gateway daemon log shows the following CTG8433E error message in the JES output STDERR, because the CICS region has released the IPIC connection, and the automatic reconnection attempt fails:

```
CTG8433E Connection failure for IPIC connection to CICS server CICSTOR1 reason =
java.net.ConnectException : EDC8128I Connection refused.
```

The updated IPConn definition can now be installed. The TCPIPService is returned to the open state using the **CEMT** command. The IPIC connection from the Gateway daemon is reestablished when one of these events occurs:

- The first ECI request to use it is processed
- The server retry interval for the Gateway daemon IPIC server definition expires

The CICS region log shows the following messages:

```
S20351T1 Number of RECEIVE sessions for IPCONN CTG0 set to 100. Number requested
100. Limit 100.
S20351T1 Client web session 1 from applid ITSOCTG0 accepted for IPCONN CTG0.
S20351T1 Client web session 2 from applid ITSOCTG0 accepted for IPCONN CTG0.
```

The Gateway daemon JES output STDOUT shows the following message:

```
CTG8429I Established new IPIC connection to CICS server CICSTOR1 with: negotiated
session limit=30, CICSAPPLID=S20351T1 CICSAPPLIDQUALIFIER=USIBMSC ,
HOSTNAME=lp01.redbooks.ibm.com, PORT=9000, sockets=2
```

10.3.2 Test using the CTGTESTR job

You can use CTGTESTR with security credentials to validate the secure IPIC connection. Example 10-17 shows where to add a user ID and password parameter in the CTGTESTR job to test the secure IPIC connection.

Example 10-17 Adding a user ID and password parameter to the CTGTESTR parameters

```
//STDENV   DD    *
PATH=/bin:/usr/lpp/java/J7.0/bin
CTGTEST_OPTS=jgate=tcp://localhost \
jgateport=2016 server=CICSTOR1 \
userid=ctguser password=ctgpass \
prog0=EC01 commarealength=18
/*
```

With a valid set of credentials, the output from CTGTESTR is much the same as shown in Example 10-14 on page 182, except that user ID and password values are not null. Example 10-18 shows an extract of the EciB2 output, when an invalid set of credentials is supplied to CTGTESTR, indicating that CICSTOR1 has performed security validation as a result of the ECI request.

Example 10-18 Extract from EciB2 for invalid security credentials

```
About to call : EC01
  COMMAREA    :
  extend_mode : 0
  LUW_token   : 0
  Commarea    :
Return code   : ECI_ERR_SECURITY_ERROR(-27)
Abend code    : null
```

This example shows that CICS has validated the security credentials provided and that the ECI request has completed with an appropriate error response.

10.4 Configure secure connections

This section explains how to enable and verify secure connectivity between these components:

- Remote mode applications and the Gateway daemon on z/OS
- The Gateway daemon and CICS TS for z/OS

The Gateway daemon and the CICS region both use RACF key rings, and resources are provided to verify secure connectivity using a batch JOB.

Having established secure connectivity to the CICS TS for z/OS region, you allow the user ID for the ECI request to be asserted by the client application by modifying the `USERAUTH` attribute of the CICS `IPConn` definition from `VERIFY` to `IDENTIFY`, which removes the need for a password credential.

This section builds on the Gateway daemon configuration that was described in the preceding sections of this chapter. It assumes that the CICS TS for z/OS region has a secure IPIC connection configured, as described in Chapter 9, “Configuring IPIC in CICS Transaction Gateway for z/OS” on page 147, using a self-signed certificate authority (CA). The signer certificate of the CA is a key component in this configuration.

Commercial CA signer certificates: These types of certificates are typically included in Secure Sockets Layer (SSL) runtime libraries and do not need to be added manually to key rings or keystores.

10.4.1 Secure the Gateway daemon

The first step in establishing secure connectivity to or from the Gateway daemon is to provide it with the necessary credentials to authenticate itself with securely connected components. In this topic, a self-signed certificate authority (CA) is used to create a personal certificate for the Gateway daemon.

Create and populate the Gateway daemon key ring

Each RACF user ID can own multiple key rings, but each key ring has a single owner. A key ring might contain multiple personal certificates, together with multiple CA signer certificates. Each certificate is identified by a label that must be unique to the user ID with which the certificate is associated. One personal certificate within the key ring is marked as the default certificate.

To enable secure connectivity for the Gateway daemon, define a key ring in the `PRODUCT` section of the configuration file. The default personal certificate within the specified key ring is selected automatically to represent the Gateway daemon.

Default certificate: There is no capability within the Gateway daemon to select a specific personal certificate from the key ring. The default certificate is always used.

A specific key ring is created for the Gateway daemon `CTGGW` started task user ID using the following command:

```
RACDCERT ADDRING(CTGGW-KEYRING) ID(CTGGW)
```

A personal certificate is created for the Gateway daemon with the label `ITSO-GATEWAY-DAEMON-CERT`. It is signed by the CA with the label `CICS-Sample-Certification`, the same CA that is used to sign the personal certificate of the CICS region that is described in Chapter 9, “Configuring IPIC in CICS Transaction Gateway for z/OS” on page 147. The personal certificate is associated with the Gateway daemon `CTGGW` started task user ID. We used the following RACF command:

```
RACDCERT ID(CTGGW) GENCERT SUBJECTSDN(OU('ITSO REDBOOKS') O('IBM') CN('ITSO
GATEWAY DAEMON CERT') C('USA')) WITHLABEL('ITSO-GATEWAY-DAEMON-CERT') SIGNWITH
(CERTAUTH LABEL('CICS-Sample-Certification'))
```

Default key pair: By default, `RACDCERT GENCERT` generates a key pair using software, with the RSA algorithm and the private key stored in the RACF database as a non-ICSF RSA key. Alternative key types are documented in the `RACFDCERT GENCERT` topic of *z/OS Security Server RACF Command Language Reference*, SA22-7687.

Add the Gateway daemon personal certificate to the key ring as the default using the following **`RACDCERT CONNECT`** command:

```
RACDCERT CONNECT(LABEL('ITSO-GATEWAY-DAEMON-CERT') RING(CTGGW-KEYRING) DEFAULT
USAGE(PERSONAL)) ID(CTGGW)
```

The CA signer certificate is required in the Gateway daemon key ring to establish a secure IPIC connection with the CICS region, as described in 10.4.3, “SSL IPIC to CICS” on page 193. Add the signer certificate to the Gateway daemon key ring, using the following **`RACDCERT CONNECT`** command:

```
RACDCERT CONNECT(CERTAUTH LABEL('CICS-Sample-Certification') RING(CTGGW-KEYRING)
USAGE(CERTAUTH)) ID(CTGGW)
```

Verify the Gateway daemon key ring

Verify the contents of the Gateway daemon key ring, using the following **`RACDCERT LISTRING`** command:

```
RACDCERT LISTRING(CTGGW-KEYRING) ID(CTGGW)
```

Example 10-19 shows the output from the **`RACDCERT LISTRING`** command, confirming the inclusion of the `CICS-Sample-Certification` signer certificate and the `ITSO-GATEWAY-DAEMON-CERT` Gateway daemon personal certificate as the key ring default certificate.

Example 10-19 Listing the details for the Gateway daemon key ring

```
Digital ring information for user CTGGW:

  Ring:
       >CTGGW-KEYRING<
  Certificate Label Name             Cert Owner     USAGE      DEFAULT
  -------------------------------    ------------   --------   -------
  CICS-Sample-Certification          CERTAUTH       CERTAUTH     NO
  ITSO-GATEWAY-DAEMON-CERT           ID(CTGGW)      PERSONAL     YES
```

Verify the key ring membership of the certificate authority

Display information related to the CA signer certificate with the `CICS-Sample-Certification` label using the following **RACDCERT CERTAUTH** command:

```
RACDCERT CERTAUTH LIST(LABEL('CICS-Sample-Certification')
```

Example 10-20 shows the output from the **RACDCERT CERTAUTH** command, including the `Ring Associations` summary.

Example 10-20 Listing the details for the certificate authority signer certificate

```
Digital certificate information for CERTAUTH:

  Label: CICS-Sample-Certification
  Certificate ID: 2QiJmZmDhZmjgcPJw+Jg4oGUl5OFYMOFmaOJhomDgaOJlpVA
  Status: TRUST
  Start Date: 2005/01/01 00:00:00
  End Date:   2024/12/31 23:59:59
  Serial Number:
       >00<
  Issuer's Name:
       >CN=CICS Sample Certification Authority.OU=CICS Test Department.O=Lead
       >ing Edge CICS Enterprise.L=Gotham City.SP=Xanadu.C=US<
  Subject's Name:
       >CN=CICS Sample Certification Authority.OU=CICS Test Department.O=Lead
       >ing Edge CICS Enterprise.L=Gotham City.SP=Xanadu.C=US<
  Key Usage: CERTSIGN
  Key Type: RSA
  Key Size: 768
  Private Key: YES
  Ring Associations:
    Ring Owner: CICSTOR1
    Ring:
       >Cics.Cicstor1<
    Ring Owner: CTGGW
    Ring:
       >CTGGW-KEYRING<
```

The `Ring Associations` summary confirms entries for the `CICS-Sample-Certification` CA signer certificate in both the `Cics.Cicstor1` CICS region key ring and the `CTGGW-KEYRING` Gateway daemon key ring.

Configure the Gateway daemon to use the new key ring

You can configure the Gateway daemon to use the `CTGGW-KEYRING` key ring in the `PRODUCT` section of the configuration file. Example 10-21 shows the `PRODUCT` section based on 10.1.3, “Configuration file” on page 174 after appending the **KeyRing** and **ESMKeyRing** parameters. Setting the `ESMKeyRing` parameter to `ON` instructs the Gateway daemon to read the `KeyRing` value as a RACF key ring, rather than as a Java keystore (`.jks`) file.

Example 10-21 Configuring the Gateway daemon with a key ring

```
SECTION PRODUCT
    Applid = ITSOCTGO # The APPLID of the Gateway daemon
    ApplidQualifier = CTGALPOO  # The APPLID qualifier of the Gateway daemon
    DefaultServer = CICSTOR1    # The default server used by the Gateway daemon
    KeyRing=CTGGW-KEYRING
```

```
    ESMKeyRing=ON
ENDSECTION
```

For remote applications to make secure connections to the Gateway daemon, an SSL protocol handler is required. Example 10-22 shows the protocol handler definition that is required in the `GATEWAY` section of the configuration file, based on 10.1.3, “Configuration file” on page 174.

Example 10-22 Configuring the Gateway daemon with an SSL protocol handler

```
# Protocol handler for SSL requests
   protocol@ssl.handler = com.ibm.ctg.server.SslHandler
   protocol@ssl.parameters = bind=;\
                             clientauth=off;\
                             connecttimeout=2000;\
                             idletimeout=600000;\
                             pingfrequency=60000;\
                             port=8070;\
                             solinger=10;
```

Verify that the Gateway daemon initializes successfully and that it is listening for *secure* application requests on the intended port number. Example 10-23 shows those key log messages from the Gateway daemon initialization, indicating these details.

Example 10-23 Verifying Gateway daemon initialization for secure connections

```
CTG8405I JSSE provider info: IBM JSSE2 7.0 Provider Implementation, IBM
Corporation, 7.0 build_20120529
CTG8489I The following cipher suites are provided by JSSE:
TLS_EMPTY_RENEGOTIATION_INFO_SCSV
    SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
    SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256
    SSL_RSA_WITH_AES_128_CBC_SHA256
    SSL_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
(many more ciphers listed)

CTG6524I Successfully started handler for the ssl: protocol on port 8070
```

10.4.2 SSL application to Gateway daemon

You can use the pre-built Java sample program, `com.ibm.ctg.samples.eci.EciB1`, to verify the secure connection to the Gateway daemon. The sample ECI program requires a Java keystore that contains the signer certificate for the Gateway daemon personal certificate. You can create the keystore and populate it using the z/OS SDK command-line utility program, **`keytool`**, in OMVS.

Export the CA signer certificate for keytool

Export the signer certificate, with the `CICS-Sample-Certification` label, to an MVS data set, and then copy it to a z/OS UNIX file using the **RACDCERT EXPORT** and **OPUT** commands. The `CICSRS4` TSO user ID, with an OMVS segment and `/u/cicsrs4` z/OS UNIX home directory, issues the following commands:

```
RACDCERT CERTAUTH EXPORT(LABEL('CICS-Sample-Certification'))
DSN('CICSRS4.CERTAUTH.CICS.B64') FORMAT(CERTB64)

OPUT 'CICSRS4.CERTAUTH.CICS.B64' '/u/cicsrs4/ctgtest/cics.ca.cer'
```

> **Important:** Do not use the `BINARY` option to the **OPUT** command. Otherwise, the z/OS UNIX file will not include important new line characters that are required for importing with the **keytool** utility.

You can visually inspect the exported signer certificate to verify that it is in the correct format. Example 10-24 shows the content of the `/u/cicsrs4/ctgtest/cics.ca.cer` z/OS UNIX file displayed in OMVS using the **cat** command.

Example 10-24 The base64-encoded signer certificate after export and copy to a z/OS UNIX file


```
-----BEGIN CERTIFICATE-----
MIIDCDCCApKgAwIBAgIBADANBgkqhkiG9w0BAQUFADCBqDELMAkGA1UEBhMCVVMx
DzANBgNVBAgTBlhhbmFkdTEUMBIGA1UEBxMLR290aGFtIENpdHkxJTAjBgNVBAoT
HExlYWRpbmcgRWRnZSBDSUNTIEVudGVycHJpc2UxHTAbBgNVBAsTFENJQ1MgVGVz
dCBEZXBhcnRtZW50MSwwKgYDVQQDEyNDSUNTIFNhbXBsZSBDZXJ0aWZpY2F0aW9u
IEF1dGhvcml0eTAeFw0wNTAxMDEwNDAwMDBaFw0yNTAxMDEwMzU5NTlaMIGoMQsw
CQYDVQQGEwJVUzEPMA0GA1UECBMGWGFuYWR1MRQwEgYDVQQHEwtHb3RoYW0gQ2l0
eTElMCMGA1UEChMcTGVhZGluZyBFZGdlIENJQ1MgRW50ZXJwcmlzZTEdMBsGA1UE
CxMUQ0lDUyBUZXN0IERlcGFydG1lbnQxLDAqBgNVBAMTIONJQ1MgU2FtcGxlIENl
cnRpZmljYXRpb24gQXV0aG9yaXR5MHwwDQYJKoZIhvcNAQEBBQADawAwAJhA0IJ
GMO5hs3+Dt6yeUl684+542wta5rg5mD6tp2cnbvNXEstrwoQZ7ZwNDRYekOEy2Mc
7Kw7bTkd/ftqVCmZRYKi4TtwJcm0+aQphSGJWv5TOOtd6DnY3HR2ENoO1ffEOQID
AQABo4GEMIGBMD8GCWCGSAGG+EIBDQQyEzBHZW5lcmFOZWQgYnkgdGhlIFNlY3Vy
aXR5IFNlcnZlciBmb3Igei9PUyAoUkFDRikwDgYDVR0PAQH/BAQDAgEGMA8GA1Ud
EwEB/wQFMAMBAf8wHQYDVR0OBBYEFJiVCIzDeLA/1aWOpygJNpBO9zGYMAOGCSqG
SIb3DQEBBQUAA2EADICu8Hmeny8cUAmfKyaJCkb313IH+QOaeKzIeEaOldVMUsYv
n6V8D3W66RwcqywqSJyhIZDEpvCo17gVEc7Lnh63cxnvN6oD/OuX3Zrp9CFOdLEJ
vNru4VKeqd/LURs1
-----END CERTIFICATE-----
```


Create and populate the client keystore

Still in OMVS, use the SDK **keytool** utility to create a new key database of type JKS, and import the CA signer certificate in a single step. You must specify a password when creating a new key database. A password is also required for any subsequent **keytool** utility operations. Example 10-25 on page 191 shows the **keytool** command that is used to create the Java keystore `/u/cicsrs4/ctgtest/eci_keyring.jks` file and the resulting output.

```
./keytool -import -file /u/cicsrs4/ctgtest/cics.ca.cer -keystore
/u/cicsrs4/ctgtest/eci_keyring.jks -storepass keyringpassw0rd
```

> **Invoking the keytool command:** The current working directory (CWD) is set to the `bin` directory of the SDK, and the **keytool** command is qualified with the z/OS UNIX shorthand path for the CWD (`./`). It is equally valid to include the SDK `bin` directory in the z/OS UNIX PATH and then invoke the **keytool** command without this prefix.

The **keytool** command displays the details of the signer certificate to be imported, and the following prompt for you to decide whether to proceed with the import:

```
Trust this certificate? [no]:
```

Responding yes to this prompt results in a confirmation that the certificate import is successful:

```
Certificate was added to keystore
```

Verify the contents of the new Java keystore, again using the **keytool** utility. Example 10-25 shows the **keytool** command, using the **list** option with the verbose (-v) switch, and the resulting console output.

Example 10-25 The keytool command to inspect the Java keystore contents

```
CICSRS4 @ SC66:/Z1DRC1/usr/lpp/java/J7.0/bin>./keytool -list -keystore
/u/cicsrs4/ctgtest/eci_keyring.jks -storepass keyringpassw0rd -v

Keystore type: jks
Keystore provider: IBMJCE

Your keystore contains 1 entry

Alias name: mykey
Creation date: Jul 4, 2013
Entry type: trustedCertEntry

Owner: CN=CICS Sample Certification Authority, OU=CICS Test Department, O=Leading
Edge CICS Enterprise, L=Gotham City, ST=Xanadu, C=US
Issuer: CN=CICS Sample Certification Authority, OU=CICS Test Department, O=Leading
Edge CICS Enterprise, L=Gotham City, ST=Xanadu, C=US
Serial number: 0
Valid from: 12/31/04 11:00 PM until: 12/31/24 10:59 PM
Certificate fingerprints:
         MD5:  A2:5E:80:74:F6:EA:51:27:CA:D7:B3:91:C9:38:6D:62
         SHA1: BA:0E:05:9B:94:13:2B:D6:54:0A:72:2B:04:41:39:40:EC:97:04:38
         SHA256:
4D:6B:76:B1:E9:80:7D:ED:86:62:C6:0B:F5:C7:CA:F9:B2:AB:12:0A:62:1C:CC:45:B5:C7:80:6
4:BB:04:14:64
         Signature algorithm name: SHA1withRSA
         Version: 3

Extensions:

#1: ObjectId: 2.16.840.1.113730.1.13 Criticality=false
(unprintable characters, likely ASCII text)
#2: ObjectId: 2.5.29.19 Criticality=true
BasicConstraints:[
CA:true
PathLen:2147483647
]

#3: ObjectId: 2.5.29.15 Criticality=true
KeyUsage [
  Key_CertSign
  Crl_Sign
```

```
]

#4: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 98 95 08 8c c3 78 b0 3f  d5 a5 8e a7 28 09 36 90  .....x........6.
0010: 74 f7 31 98                                        t.1.
]
]
```

This example shows that the keystore `/u/cicsrs4/ctgtest/eci_keyring.jks` file is of type `JKS`, and the `Extensions` section of the single certificate entry includes the text `CA:true`, indicating that this is a CA signer certificate.

This keystore is used by a Java client application (or JCA connection factory) to establish a secure connection with the Gateway daemon. The presence of the CA signer certificate enables the client application SSL runtime libraries to authenticate the Gateway daemon personal certificate that is sent by the Gateway daemon as part of the SSL handshake process.

Verify the secure application connection

The `CTGTESTR` does not provide the facility to specify a client keystore for use in establishing an SSL connection to the Gateway daemon. Alternatively, the `com.ibm.ctg.samples.eci.EciB1` (`EciB1`) Java sample does, and it is supplied precompiled in the CICS TG product `classes/ctgsamples.jar` file.

Sample script to run the EciB1 Java sample program: You can run the `EciB1` CICS TG Java sample program manually from OMVS using the standard **`Java`** launcher command. You can use a sample z/OS UNIX shell script (**`ecib1`**), which is included with this book, to allow the Java `EciB1` sample to be invoked with keystore parameters through a JOB. The full details of this script and JOB are included in Appendix A, “Running the precompiled Java sample EciB1 from batch on z/OS” on page 359.

Example 10-26 shows the output from the `EciB1` Java sample program, when provided with the parameters:

`ssl://lp01.redbooks.ibm.com 8070 /u/cicsrs4/ctgtest/eci_keyring.jks keyringpassw0rd`

Example 10-26 Using the EciB1 Java sample program to verify secure connectivity

```
ecib1: Executing EciB1 with parameters:  ssl://lp01.redbooks.ibm.com 8070
/u/cicsrs4/ctgtest/eci_keyring.jks keyringpassw0rd
ecib1: Classpath=
/usr/lpp/cicstg/ctg900/classes/ctgsamples.jar:/usr/lpp/cicstg/ctg900/classes/ctgcl
ient.jar
CICS Transaction Gateway Basic ECI Sample 1
Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway URL]
                                          [Gateway port number]
                                          [SSL keyring]
                                          [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://lp01.redbooks.ibm.com port:8070.
CICS servers defined:
```

```
         1. CICSTOR1 -IPIC Server
Choose server to connect to, or q to quit:
Program ECO1 returned with data:-
        Hex: 30332f30372f31332031303a33333a34320
        ASCII text: 03/07/13 10:33:42
```

The Gateway URL, `ssl://lp01.redbooks.ibm.com`, and port number, `8070`, indicate that `EciB1` will establish a secure connection to the Gateway daemon listening on port 8070, for TCP/IP host `lp01.redbooks.ibm.com`. We must use either the host name for the static virtual IP address (VIPA), or the IP address, 192.168.0.1, rather than localhost, because the Gateway daemon protocol handler is explicitly bound to that address.

CICS server selection: Running on OMVS, the user is prompted to select a CICS server. The **`ecib1`** script automatically provides the keyboard input of `1`, which selects the first CICS server available.

The client application has established a secure connection to the Gateway daemon and has flowed an ECI request for CICS program `ECO1`. However, the IPIC connection used is still not secured by SSL.

10.4.3 SSL IPIC to CICS

Although the Gateway daemon and the CICS region described in this scenario are colocated within the same LPAR, some users regard such a configuration as inherently secure, because local adapter connections can be optimized on z/OS through the Fast Local Sockets facility. However, a Gateway daemon and a CICS region connected by IPIC might not be colocated, and so securing such a connection is a valid requirement.

In this topic, the IPIC connection to the CICS TS for z/OS region described in “IPIC connection” on page 176 is modified to use SSL. The Gateway daemon RACF key ring configured in 10.4.1, “Secure the Gateway daemon” on page 186 for use by secure remote application connections is also used for secure IPIC connections to CICS. The CA signer certificate is required at run time to authenticate the identity of the CICS region.

To secure the IPIC connection, changes are required to the `TCPIPService` definition in the target CICS region and to the IPIC Server definition in the Gateway daemon configuration file.

Configure the CICS TCPIPService for IPIC SSL

Chapter 9, “Configuring IPIC in CICS Transaction Gateway for z/OS” on page 147 includes configuration of secure IPIC connections in CICS TS for z/OS. This section highlights the minimum requirements for server authentication and the use of a common CA signer certificate.

The CICS TS for z/OS `TCPIPService` definition includes the **`SSL`** parameter, with possible values of `Yes`, `No`, or `Clientauth`. Predefined and auto-installed IPIC connections to a Gateway daemon use secure connections when the associated `TCPIPService` includes an **`SSL`** parameter of `Yes` or `Clientauth`.

Note: CICS TS for z/OS `TCPIPService` and `IPConn` resource types both include an `SSl` attribute, with possible values of `Yes`, `No`, or `Clientauth`, and a `Ciphers` attribute. Only the `TCPIPService` attribute values are relevant to IPIC connections with CICS TG components (Gateway daemon, or a local mode application).

At run time, the `IPConn` resource shows the `SSltype` attribute with a value `Nossl`, regardless of whether the IPIC connection to the CICS TG component is using SSL.

Example 10-27 shows the SECURITY section of a `TCPIPService` definition, where the `SSl` attribute value is set to `Yes` and the `CIphers` attribute is the default set for STRONG encryption. The certificate with the `CICSTOR1-ssl` label is signed by the same CA signer certificate, `CICS-Sample-Certification`, which is used to sign the Gateway daemon personal personal certificate in “Create and populate the Gateway daemon key ring” on page 186.

Example 10-27 Enabling SSL on an IPIC TCPIPService definition

```
SECURITY
  SSl            : Yes                Yes | No | Clientauth
  CErtificate    : CICSTOR1-ssl
  (Mixed Case)
  PRIvacy        : Supported          Notsupported | Required | Supported
  CIphers        : 050435363738392F303132330A1613100D0915120F0C0306
  AUthenticate   :                    No | Basic | Certificate | AUTORegister
                                      | AUTOMatic
  Realm          :
 (Mixed Case)
 ATtachsec      :                    Local | Verify
```

Configure the Gateway IPIC Server for SSL

The IPIC Server definition in the Gateway daemon configuration file includes a parameter to control the use of SSL. Example 10-28 shows the IPIC Server configuration that was previously defined in “IPIC connection” on page 176 modified to enable the use of SSL. The CICS `TCPIPService` `SSl` attribute determines whether server or client authentication is used.

Example 10-28 An IPIC server definition with SSL enabled

```
SECTION IPICSERVER = CICSTOR1
   Description = IPIC Server
   Hostname = lp01.redbooks.ibm.com
   Port = 9001
   TcpKeepAlive = Y
   Ssl = Y
   CICSApplid = S20351T1
   CICSApplidQualifier = USIBMSC
   SendSessions = 50
ENDSECTION
```

The **`Ssl`** parameter in the IPIC server definition can be set to `Y` (yes) or `N` (no). The Gateway daemon key ring, described in 10.4.1, “Secure the Gateway daemon” on page 186, contains the certificates used for server authentication and client authentication.

Verify the secure CICS connection

Gateway daemon log messages, CICS log messages, and CICS resource inquiry commands do not reveal any detail to indicate when an IPIC connection is using SSL. However, the CICS TG statistic `CSx_PROTOCOL`, where *x* is the name of the IPIC Server definition, indicates a value of either `IPIC`' or `IPICSSL`. This statistic can be inspected from TSO, using the modify command, or from the CICS connections view of the CICS TG perspective of the CICS Explorer.

Example 10-29 shows the modify command and sample output from the command:

```
/F CTGGWX.GWO,APPL=STATS,GS=CSCICSTOR1, ST=S
```

Example 10-29 Display startup statistics for CICS server CICSTOR1

```
CTG8239I Response received from CICS Transaction Gateway
CSCICSTOR1 - CICS Server (instance)
   CSCICSTOR1_SPROTOCOL=IPICSSL (CICS server protocol)
   CSCICSTOR1_SIPADDR=lp01.redbooks.ibm.com (CICS server TCP/IP
address)
   CSCICSTOR1_SIPPORT=9001 (CICS server TCP/IP port)
   CSCICSTOR1_SSESSMAX=100 (Number of requested IPIC sessions)
```

The secure connection is tested using the `EciB1` sample, as described in “Verify the secure application connection” on page 192. The input parameters and output appear unchanged, compared to Example 10-26 on page 192. Securing the IPIC connection from the Gateway daemon to the CICS server with SSL is a transparent change for remote-mode applications.

Asserted identity using the secure CICS connection

After you verify the secure IPIC connection, you can change the `USERAUTH` attribute of the IPConn definition from `VERIFY` to `IDENTIFY`. Then, repeat the test using the `EciB1` sample, as described in “Verify the secure application connection” on page 192, but this time, omit the password credential.

Note: `USERAUTH(IDENTIFY)` is always permitted by CICS TS for z/OS when the IPIC connection uses SSL, regardless of the location or platform of the Gateway daemon. It is also permitted for non-SSL IPIC connections, when the Gateway daemon is on an MVS image within the same sysplex as the CICS TS for z/OS region.

Although it is not typical to secure intra-sysplex connections between Gateway daemons and CICS regions, it is sometimes required. It suits the purpose of this chapter, by illustrating the configuration of a secure IPIC connection.

10.4.4 Restricting cipher suites

During the establishment of a secure connection, the intersection of cipher suites that are available to both parties defines the set of the potential algorithms available for use. It is possible to explicitly define independent sets of cipher suites for use between the following components:

- Remote-mode applications and the Gateway daemon
- The Gateway daemon and the CICS server (using IPIC)

Cipher suites for remote-mode application connections

You can restrict the cipher suites that are available for use between remote-mode applications and the Gateway daemon using the `ciphersuites` attribute of the SSL protocol handler in the `GATEWAY` section of the configuration file. Example 10-30 shows the SSL protocol handler definition from "Configure the Gateway daemon to use the new key ring" on page 188, appended with a restricted set of cipher suites.

Example 10-30 Configuring a restricted set of cipher suites for remote-mode applications

```
# Protocol handler for SSL requests
   protocol@ssl.handler = com.ibm.ctg.server.SslHandler
   protocol@ssl.parameters = bind=;\
       clientauth=off;\
       connecttimeout=2000;\
       idletimeout=600000;\
       pingfrequency=60000;\
       port=8070;\
       solinger=10;\
       ciphersuites=\
       SSL_ECDHE_RSA_WITH_RC4_128_SHA,\
       SSL_RSA_WITH_RC4_128_SHA,\
       SSL_ECDH_ECDSA_WITH_RC4_128_SHA,\
       SSL_ECDH_RSA_WITH_RC4_128_SHA;
```

The `CTG8489I` Gateway daemon initialization log message lists all of the available cipher suites. The `CTG8401I` log message shown in Example 10-31 lists the restricted cipher suites that are configured for use by the SSL protocol handler.

Example 10-31 Verifying a restricted set of cipher suites available for remote-mode applications

```
CTG8401I The following cipher suites are available for the SSL protocol handler:
 SSL_ECDH_RSA_WITH_RC4_128_SHA
 SSL_RSA_WITH_RC4_128_SHA
 SSL_ECDH_ECDSA_WITH_RC4_128_SHA
 SSL_ECDHE_RSA_WITH_RC4_128_SHA
```

You can determine the specific cipher suite by a particular remote-mode application by enabling connection logging in the Gateway daemon using the following configuration option:

```
ConnectionLogging = on      # Log clients connecting and disconnecting
```

Example 10-32 shows a pair of log messages that correspond to the secure connection of a client application, and subsequent disconnection from a client application. Both log messages include details of the protocol (`TLSv1`) and the cipher suite (`SSL_ECDHE_RSA_WITH_RC4_128_SHA`).

Example 10-32 Connection logging message showing specific cipher suite in use

```
CTG6506I Client connected: [ConnectionManager-99] - ssl@a6268494
[SSL_ECDHE_RSA_WITH_RC4_128_SHA:
Socket[addr=/127.0.0.1,port=40342,localport=8070]] using protocol TLSv1

CTG6507I Client disconnected: [ConnectionManager-99] - ssl@a6268494
[SSL_ECDHE_RSA_WITH_RC4_128_SHA: Socket
[addr=/127.0.0.1,port=40342,localport=8070]] using protocol TLSv1, reason = Java
client application closed the connection
```

Cipher suites for IPIC connections to CICS servers

The Gateway daemon configuration allows independent control over the cipher suites for each IPIC SSL connection definition. The configuration is similar to specifying cipher suites for the SSL protocol handler, where a comma-separated list defines the set of cipher suites to allow. Similar to the SSL protocol handler definition, line-continuation characters, such as a backslash (\), are used for legibility, as shown in Example 10-33.

Example 10-33 Configuring a restricted set of cipher suites for an IPIC connection

```
SECTION IPICSERVER = CICSTOR1
   Description = IPIC Server
   Hostname = lp01.redbooks.ibm.com
   Port = 9001
   TcpKeepAlive = Y
   Ssl = Y
   CICSApplid = S20351T1
   CICSApplidQualifier = USIBMSC
   SendSessions = 100
   CipherSuites=\
     SSL_RSA_WITH_NULL_MD5,\
     SSL_RSA_WITH_DES_CBC_SHA,\
     SSL_DHE_RSA_WITH_DES_CBC_SHA,\
     SSL_DHE_DSS_WITH_DES_CBC_SHA,\
     SSL_DH_anon_WITH_DES_CBC_SHA,\
     SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA,\
     SSL_RSA_FIPS_WITH_DES_CBC_SHA
ENDSECTION
```

The IPIC connection is established when the first ECI request specifying the CICS server name is received. Example 10-34 shows the log messages, which include the restricted set of cipher suites in use.

Example 10-34 Verifying a restricted set of cipher suites available for remote-mode applications

```
CTG8477I About to connect to server CICSTOR1
CTG8488I The following cipher suites are enabled for IPIC server CICSTOR1:
 SSL_RSA_FIPS_WITH_DES_CBC_SHA
 SSL_RSA_WITH_DES_CBC_SHA
 SSL_DH_anon_WITH_DES_CBC_SHA
 SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
 SSL_DHE_DSS_WITH_DES_CBC_SHA
 SSL_RSA_WITH_NULL_MD5
 SSL_DHE_RSA_WITH_DES_CBC_SHA
```

Relationship between the cipher suites and Gateway daemon key ring

The personal certificate for the Gateway daemon is created using a specific key type, for example, RSA. The selected key type imposes a restriction of its own on the possible set of cipher suites to use at run time by the Gateway daemon. In “Create and populate the Gateway daemon key ring” on page 186, the default key type of RSA was used. Therefore, any restricted set of cipher suites to be used with this personal certificate must include at least one based on the RSA algorithm.

If an SSL connection (SSL protocol handler or IPIC SERVER) is restricted to a set of cipher suites without support for the key type of the Gateway daemon personal certificate, the connection will fail during the SSL handshake.

SP800-131A compliance

Compliance with the National Institute of Standards and Technology (NIST) SP800-131A standard is achieved through the Java system properties for the IBM Java Secure Socket Extension (JSSE) runtime library, `com.ibm.jsse2.sp800-131`. Example 10-35 shows how to use the **`CTGSTART_OPTS`** environment variable to restrict encryption algorithms to the "transition" set. This setting requires all SSL connections with the Gateway daemon to support at least Transport Layer Security (TLS) 1.0. The "strict" set requires all SSL connections with the Gateway daemon to support at least TLS 1.2.

Example 10-35 Configuring the Gateway daemon for SP800-131A compliance through STDENV

```
CTGSTART_OPTS=-j-Dcom.ibm.jsse2.sp800-131=transition
```

The same IBM JSSE property is also used with Java client applications to ensure the use of stronger encryption algorithms. For further details, see the CICS TG Information Center *SP800-131A compliance* topic:

http://pic.dhe.ibm.com/infocenter/cicstgmp/v9r0/index.jsp

Full SP800-131 compliance (`strict`) is available for the CICS TG V9.0 products by applying the following maintenance:

- CICS TG for z/OS V9.0 APAR PM98779
- CICS TG for Multiplatforms V9.0.0.2 (V9.0 Fix Pack 2)
- CICS TG Desktop Edition V9.0.0.2 (V9.0 Fix Pack 2)

TLS 1.2 IPIC connectivity is available for CICS Transaction Server for z/OS V5.1 by applying the SP800-131A APAR PM97207.

10.5 Adding XA support

To enable the Gateway daemon for XA transaction support, the Gateway daemon started task user ID requires the correct security permissions, and further system configuration is required to successfully start the non-terminating address space, `CTGRRMS`.

10.5.1 The CTGRRMS address space

The `CTGRRMS` address space provides a proxy service within the scope of a single MVS image to RRMS capabilities for Gateway daemons requiring XA transaction support. It is a special type of address space, known as a *non-terminating address space*, that does not consume CPU beyond initialization and requires a minimal amount of storage.

Each LPAR runs a maximum of one `CTGRRMS` address space. When at least one Gateway daemon on an LPAR requires XA transaction support, a `CTGRRMS` address space is required to run on that LPAR.

You can start the CTGRRMS address space using one of the following methods:

- Implicit start:
  - During initialization of a Gateway daemon with XA transaction support, the CTGRRMS address space is started, if it is not already running.
  - Typically, the first Gateway daemon with XA transaction support to initialize after an IPL of the z/OS LPAR triggers the start of the CTGRRMS address space.
- Explicit start

  A system administrator uses the **ctgasi** z/OS UNIX command to explicitly start the CTGRRMS address space.

An implicit start of CTGRRMS is the usual pattern of operation. Explicit lifecycle operations, such as start, stop, or restart using the **ctgasi** command, can be required to apply maintenance, or during problem determination. However, when configured correctly, the start and use of CTGRRMS is transparent to the Gateway daemon administrator.

10.5.2 System configuration for CTGRRMS

The single CICS TG product load-module used by the CTGRRMS address space is CTGINIT, which is in the product load library, SCTGLINK. The SCTGLINK load library must have the following characteristics:

- Defined as authorized program facility (APF)-authorized
- Included in the MVS library lookaside (LLA) address space

The Gateway daemon started task user ID or the system administrator user ID must have at least UPDATE level access to the CTG.RRMS.SERVICE in the FACILITY class to start CTGRRMS.

APF authorization

The CICS TG load library, SCTGLINK, is made APF-authorized by adding it to the APF list by using the **SETPROG APF** command:

```
SETPROG APF,ADD,DSNAME=CTG.V9R0M0.SCTGAUTH,VOLUME=TST00D
```

Making dynamic updates: You can use the **SETPROG APF** command to make dynamic updates to the APF-authorized status of a load library. To make the APF authorization persistent, include an APF ADD statement in a system PARMLIB(PROGxx) member:

```
APF ADD
    DSNAME(CTG.V9R0M0.SCTGLINK)                          VOLUME(TST00D)
```

You can verify the APF-authorized status of load libraries by using the MVS display **/D PROG,APF** command. For the examples in this book, multiple versions of CICS TG for z/OS are installed on the system, and all systems share a common high-level qualifier, CTG.

Example 10-36 shows the output from the MVS display command, when the DSNAME option is used to summarize the APF authorization of CICS TG product load libraries on the LP01 LPAR.

*Example 10-36 Output from MVS command /D PROG,APF,DSN=CTG**

```
RESPONSE=LP01
 CSV450I 07.06.15 PROG,APF DISPLAY 031
 FORMAT=DYNAMIC
 ENTRY VOLUME DSNAME
    1  TST021 CTG.V6R1M0.SCTGAUTH
    2  TST025 CTG.V7R0M0.SCTGAUTH
    3  TST025 CTG.V7R0M0.SCTGLINK
    4  TST029 CTG.V7R1M0.SCTGAUTH
    5  TST029 CTG.V7R1M0.SCTGLINK
    6  TST00D CTG.V8R0M0.SCTGAUTH
    7  TST00D CTG.V8R0M0.SCTGLINK
    8  TST00D CTG.V9R0M0.SCTGAUTH
    9  TST00D CTG.V9R0M0.SCTGLINK
```

Implementing automatic restart manager: The `SCTGAUTH` load library is not relevant to XA transaction support but it needs to be APF-authorized, if required. This load library includes the `CTGARM` utility program, which is used for implementing automatic restart manager (ARM) support with a Gateway daemon. For more information, see the CICS TG for z/OS Information Center *Automatic restart management* topic:

http://pic.dhe.ibm.com/infocenter/cicstgmp/v9r0/index.jsp

Library lookaside (LLA) address space

The CICS TG load library, `SCTGLINK`, is added to the library lookaside address space, alternatively known as the *LNKLST*, by using the following **`SETPROG LNKLST`** commands:

```
SETPROG LNKLST,DEFINE,NAME=T1,COPYFROM=CURRENT
SETPROG LNKLST,ADD,NAME=T1,DSNAME=CTG.V9R0M0.SCTGLINK,
VOLUME=TST00D
SETPROG LNKLST,ACTIVATE,NAME=T1
```

The `LNKLST` name `T1` is used as a temporary location to hold a concatenation of the currently active `LNKLST` and the `SCTGLINK` load library until the **`SET PROG LNKLST,ACTIVATE`** command is issued.

Making harden persist changes: You can use the **`SETPROG LNKLST`** commands to make dynamic updates to the LLA data set concatenation. However, to harden persist these changes, add similar commands to the system `PARMLIB(PROGxx)` member:

```
LNKLST DEFINE NAME(LNKLSTT1) COPYFROM(CURRENT)
LNKLST ADD    NAME(LNKLSTT1) DSNAME(CTG.V9R0M0.SCTGLINK)
VOLUME=TST00D
LNKLST ACTIVATE NAME(LNKLSTT1)
```

You can verify the contents and order of entries in the `LLA LNKLST` by using the following command:

```
D PROG,LNKLST
```

For clarity, include only one `SCTGLINK` load library entry on the `LLA LNKLIST`.

Multiple releases of CICS TG for z/OS and CTGRRMS

When multiple releases of CICS TG for z/OS are available within the same z/OS LPAR, `CTGRRMS` must use the highest available version of `CTGINIT` to ensure compatibility with all releases of the Gateway daemon that might run on the LPAR. Back-level Gateway daemons are compatible with higher-level versions of `CTGRRMS`, unless explicitly stated in the migration documentation for a specific release.

For CICS system programmers, who are familiar with the `DFHIRP` load library, the `CTGINIT` load module follows similar concepts in terms of system configuration.

> **Compatibility note:** As of CICS TG V9.0, `CTGINIT` is backwardly compatible with Gateway daemons of all previous releases.

For the full details about configuration of XA transaction support, search for the CICS TG for z/OS Information Center *Configuring XA support* topic:

http://pic.dhe.ibm.com/infocenter/cicstgmp/v9r0/index.jsp

10.5.3 Gateway daemon configuration for XA transactions

XA transaction support is enabled in the Gateway daemon through the **`xasupport`** parameter, in the `GATEWAY` section of the Gateway daemon configuration file:

```
xasupport=on
```

APPLID qualifier and APPLID parameters

Setting the APPLID qualifier and APPLID parameters for a Gateway daemon is a preferred practice, because they provide a structured naming convention to identify each Gateway daemon as a communication node. They can be useful for tracking requests, monitoring, or general problem determination.

When XA transaction support is enabled, setting the APPLID qualifier and APPLID parameters becomes mandatory. Peer-recovery of XA transactions within a highly available Gateway group requires peer-Gateway daemons to share a common APPLID qualifier.

The `PRODUCT` section of the Gateway daemon configuration file, shown in “PRODUCT section” on page 175, contains this information:

```
Applid = ITSOCTGO # The APPLID of the Gateway daemon
ApplidQualifier = CTGALPOO  # The APPLID qualifier of the Gateway daemon
```

The Gateway daemon APPLID qualifier and APPLID are used for these functions:

- To generate a Recovery Manager name for registration with Resource Recovery Services (RRS), shown in the Gateway daemon log message `CTG9295I`:

  ```
  CTG9295I The Gateway daemon is running with resource manager name
     'CICSTG.CTGALPOO.ITSOCTGO'
  ```

- To identify the Gateway daemon as an IPIC partner, shown in the CICS region log message `DFHIS3000`:

  ```
  DFHIS3000 07/08/2013 12:56:36 CICSTOR1 IPCONN CTGO with applid
     CTGALPOO.ITSOCTGO autoinstalled successfully using autoinstall user program
  DFHISAIP
     and template (NONE) after a connection request was received on tcpipservice
  GW1IPIC from
     host 192.168.0.1.
  ```

- To identify the Gateway daemon is monitoring and statistics:

```
GD_SAPPLID=ITSOCTG0 (CICS TG APPLID)
GD_SAPPLIDQ=CTGALP00 (CICS TG APPLID qualifier)
```

Cold start

A failure during distributed-sync point processing can lead to an XA transaction entering the in-doubt state. XA transactions are represented in RRS by *units of recovery* (URs). Incomplete URs persist until they are completed due to resynchronization (for example, driven by the Java EE application server), or manual intervention through RRS panels or CICS systems administration.

During initialization, the Gateway daemon retrieves the list of all incomplete URs associated with itself (from a previous run). Gateway daemon log message `CTG8628I` includes details regarding the number of URs retrieved from RRS:

```
CTG8628I Retrieved interest in 0 units of recovery from RRS; 0 are in-doubt
```

The difference between the number of retrieved interests, and the (equal, or lower) number of in-doubt URs, is the number of incomplete URs that can be cleared by a cold start of the Gateway daemon.

By default, all incomplete URs associated with a Gateway daemon Recovery Manager name persist through a restart of the Gateway daemon, and they are reported by the `CTG8628I` log message.

When the Gateway daemon start type is `COLD`, those incomplete but not in-doubt URs are completed. This is achieved with a **`ctgstart`** option "**`-start`**`=cold`" (using the **`CTGSTART_OPTS`** environment variable), or using the **`Start`** parameter in the `GATEWAY` section of the Gateway daemon configuration file:

```
Start = cold                # Clear URs that are in-forget when starting
```

> **Note:** Normal start is the default and is performed when no **`start`** parameter value is specified. The only acceptable value that you can specify with the **`start`** parameter is "`cold`".

The Gateway daemon initialization log message `CTG8986I` indicates when the start type is COLD:

```
CTG8986I CICS Transaction Gateway start type is COLD
```

Maximum concurrent XA transactions

The Gateway daemon tracks both in-flight and incomplete XA transactions, for which it has an interest. This number can be greater than the number of concurrent requests to CICS (`MAXWORKER`), and greater than the maximum connected client applications (`MAXCONNECT`), due to the inclusion of incomplete XA transactions.

The **`CTG_XA_MAX_TRAN`** environment variable controls the total number of XA transactions that a single Gateway daemon can track at any one time and has a default value of `1024`. Although the value of the **`MAXCONNECT`** parameter represents a reasonable basis for **`CTG_XA_MAX_TRAN`**, exceptional circumstances might result in a large number of incomplete XA transactions within a short time. Therefore, it is advisable to overestimate for this upper limit.

The Gateway daemon storage that is associated with tracking each XA transaction is approximately 200 bytes. For example, setting the **`CTG_XA_MAX_TRAN`** parameter to `8192` requires approximately 1.6 MB of storage.

10.5.4 Verifying the configuration for XA transactions

Now, you can verify that the Gateway daemon, the `CTGRRMS` address space, and RRS are initialized.

Because XA transaction support is used only by enterprise applications that are running in a Java EE-certified application server, you cannot verify the capability for XA transaction support using a stand-alone test program. For details about deploying and using the CICS TG-supplied JCA installation verification test (`IVT`) enterprise application to fully verify XA transaction support, see 12.7.1, “Deploying and configuring the IVT” on page 287.

Verify the Gateway daemon support for XA transactions

Example 10-37 shows the Gateway daemon initialization log messages that are related to XA transaction support.

Example 10-37 Gateway daemon initialization log messages related to XA transaction support

```
CTG8426I The APPLID is ITSOCTGO
CTG8427I The APPLID qualifier is CTGALPOO
CTG6737I XA transaction support is enabled
CTG9295I The Gateway daemon is running with resource manager name
   'CICSTG.CTGALPOO.ITSOCTGO'
CTG8628I Retrieved interest in 0 units of recovery from RRS; 0 are in-doubt
```

You can use the resource manager name that is indicated by the `CTG9295I` log message to investigate the status of extended LUW and XA transactions through the RRS menus.

Confirm the status of the CTGRRMS address space

If not already running, a Gateway daemon that is enabled for XA transactions starts the `CTGRRMS` address space. Using the System Display and Search Facility (SDSF) Display Active (DA) option, a single `CTGRRMS` address space per LPAR can be shown. However, there is no JES output from this address space and, as shown in Example 10-38, no entries in the `ProcStep`, `JobID`, or `Owner` DA columns.

Example 10-38 An extract from the SDSF DA panel showing the CTGRRMS address space

```
JOBNAME  StepName ProcStep JobID    Owner    C Pos DP Real
CTGRRMS  CTGRRMS                               NS  FF   84
```

Register the relationship with RRMS

When initialized with XA transaction support, a Gateway daemon becomes a *resource manager* with RRS.

> **Terminology note:** By contrast, when XA transaction support is not enabled, a Gateway daemon becomes a *work manager* with RRS.

From the first RRS panels, selecting option 2, Display/Update RRS-related Resource Manager information, and entering the RM name indicated by the Gateway daemon log message `CTG9295I` returns the RRS Resource Manager List panel.

Example 10-39 shows the Gateway daemon in the `Run` state, which indicates a successful initialization and registration with RRS.

Example 10-39 The RRS Resource Manager List showing the Gateway daemon status

```
                          RRS Resource Manager List            Row 1 to 1 of 1
Command ===>                                                 Scroll ===> PAGE

Commands: v-View Details u-View URs r-Remove Interest d-Delete RM
          n-Unregister RM

S   RM Name                          State             System   Logging Group
    CICSTG.CTGALP00.ITSOCTG0         Run               LP01     ITSOLP00
```

From the RRS Resource Manager List panel, you can inquire about the status of the Gateway daemon as a Resource Manager and individual URs associated with the Gateway daemon.

11

CICS Transaction Gateway Multiplatforms and Desktop Edition configuration

This chapter explains the installation and configuration of CICS Transaction Gateway (CICS TG) for Multiplatforms, using Windows for illustration and citing any major differences for UNIX/Linux platforms. The information provided also applies to the CICS TG Desktop Edition product, and the limitations of that product are highlighted, where applicable. The examples that are provided use a remote mode configuration and include end-to-end secure connectivity in a 3-tier system.

This chapter includes the following sections:

11.1 Installing CICS TG

As mentioned previously in 3.1, “Preparing to install CICS TG Multiplatforms and CICS TG Desktop Edition in a distributed environment” on page 34, the installation image consists of two files: the installer file and a readme file. This section works with the installer file. For the purpose of this book, Windows Server 2012 and AIX 7.1 were used.

11.1.1 Installing on a Windows platform

CICS TG uses the InstallAnywhere installation manager. To start the installation, download the product archive file, or `eImage`, to a temporary directory. Open the directory to verify the contents. Product installation (and uninstallation) steps must be performed by a user with Administrator authority.

The installation can be done in one of the following modes. The default mode is GUI.

- A *GUI mode installation* can be launched using the following command:

  `installer`

- A *console mode installation* can be launched using the following command:

  `installer -i console`

- An *unattended installation* can be launched using the following command:

  `installer -i silent -DLICENSE_ACCEPTED=true`

If the installation is successful, the following message displays:

```
Installation: Successful.
```

You can find the installation logs in the following directory:

```
C:\Program Files (x86)\IBM\CICS Transaction Gateway\installlogs
```

11.1.2 Installing on a UNIX (AIX) platform

CICS TG uses the InstallAnywhere installation manager. To start the installation, download the eImage to a temporary directory. Open the directory to verify the contents.

For UNIX platforms, the CICS TG installation process requires that the Korn shell is installed on the system.

The installation of the CICS TG product must be performed by a user with root authority.

Before you install the CICS TG product, check the `umask` setting of the installing user to ensure that files created during the installation are readable by the users that will run CICS TG. A `umask` setting of `077` restricts access to the user who installed CICS TG; a `umask` setting of `022` allows all users to run CICS TG.

The installation can be done in one of the following modes. The default mode is GUI.

- A *GUI mode installation* can be launched using the following command:

  installer

- A *console mode installation* can be launched using the following command:

  installer -i console

- An *unattended installation* can be launched using the following command:

  installer -i silent -DLICENSE_ACCEPTED=true

If the installation is successful, the following message displays:

```
Installation: Successful.
```

You can find the installation logs in the `/opt/IBM/cicstg/installlogs` directory.

11.2 Configure the Gateway daemon

This section describes the steps to configure the Gateway daemon for IPIC connectivity with the CICS region, `CICSTOR1`, based on the definitions described in Chapter 9, “Configuring IPIC in CICS Transaction Gateway for z/OS” on page 147. The CICS TG configuration is equally valid for connection with a TXSeries region with an equivalent IPIC connection definition.

Now that CICS TG is installed, you need to configure it by creating a configuration file (`ctg.ini`). To create a new configuration, either use the Configuration Tool or edit a copy of the sample configuration file supplied in the product sample configuration directory:

`<install_path>/samples/configuration/ctgsamp.ini`

> **Default values:** The sample configuration file `ctgsamp.ini` specifies the default value of each configuration attribute, where appropriate.

11.2.1 System-level configuration

This section describes the configuration steps required to be performed at the system level before you create a CICS TG configuration file.

Location of the configuration file

CICS TG runtime components rely on a convention to locate the CICS TG configuration file. Either the configuration file is in a default location or the location is defined by the environment variable **CICSCLI**. The convention used for the location of the configuration file uses the following sequence:

1. The location of the configuration file is defined by the **CICSCLI** environment variable.
2. If the **CICSCLI** environment variable is not defined, load the configuration file `ctg.ini` from the default location:
   - The `<product_data_path>` directory on Windows platforms, using the default file name:

     `c:\ProgramData\IBM\CICS Transaction Data\ctg.ini`

   - The `<install_path>/bin` directory on UNIX systems, using the default file name:

     `/opt/ibm/cicstg/bin/ctg.ini`

If the **CICSCLI** environment variable is defined, the configuration tool will automatically attempt to load from, and save to, the defined location and file name. Otherwise, the configuration tool will automatically use the default location and file name.

The CICS TG configuration tool can save the configuration file to a specific file name and location, independent of the **CICSCLI** environment variable and the default location. To do this, use the menu option, **File** → **Save as**.

The Gateway daemon and Client daemon components load their configurations by following the same convention.

Setting the CICSCLI environment variable

For Windows systems, the **CICSCLI** environment variable must be set as a system variable to be used by the IBM CICS Transaction Gateway service. Figure 11-1 shows the new CICSCLI system variable created by going to **Control Panel** → **System** → **Advanced system settings** → **Environment variables** and then by selecting **New**.

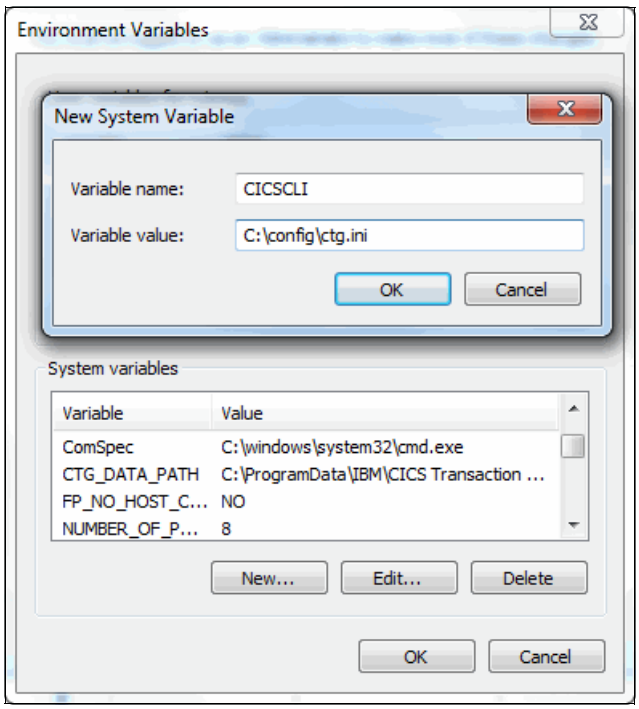


Figure 11-1 Setting the CICSCLI system variable on Windows

Note: Setting a system variable on Windows requires Administrator privileges.

For UNIX and Linux systems, the **CICSCLI** environment variable can be set by various means. For production systems, it is suggested to set the **CICSCLI** environment variable using the configuration file for the CICS TG daemon script, **ctgd**, as described in "Using the ctgd script" on page 225.

For development and test systems, the **CICSCLI** environment variable can be set in the current shell using the **export** command:

```
export CICSCLI=/etc/cicstg/cicstg.ini
```

11.2.2 Gateway daemon configuration summary

To start using CICS TG, the majority of the configuration file parameters can be left at their default values. However, to use CICS TG in remote mode with IPIC connectivity, the minimum suggested configuration that we define includes these items:

- The `PRODUCT` section, where we need to identify the Gateway daemon to CICS on the network for administrative and audit purposes
- The `GATEWAY` section. For this section, set the following parameters:
  - **`adminport`**: Confirm that this port is not already in use on your system.
  - **`protocol@tcp.handler`** and **`protocol@tcp.parameters`**: Enable the Gateway daemon to accept requests using the TCP protocol and select the port on which it listens.
  - One `IPICSERVER` section, where we need to define at least one CICS server.

Other parameters can be set to further customize your CICS TG configuration, including configuration of the Client daemon when the CICS Family TCPIP protocol or Systems Network Architecture (SNA) Advanced Program-to-Program Communication (APPC) connectivity is required. The configuration described here is based on IPIC connectivity, but 11.6, “Client daemon configuration” on page 255 uses a CICS Family TCPIP protocol connection to show how a Client daemon connection is configured.

The following sections describe the steps required to configure the Gateway daemon to perform these tasks:

- Enable the `tcp` and `statsapi` protocol handlers for the Gateway daemon.
- Define an Internet Protocol interconnectivity (IPIC) connection to a CICS Transaction Server for z/OS (CICS TS) server.

11.2.3 The PRODUCT section

The `PRODUCT` section of the configuration file defines product-wide settings. Figure 11-2 shows the CICS Transaction Gateway panel of the configuration tool that is used to generate the `PRODUCT` section of the configuration file.

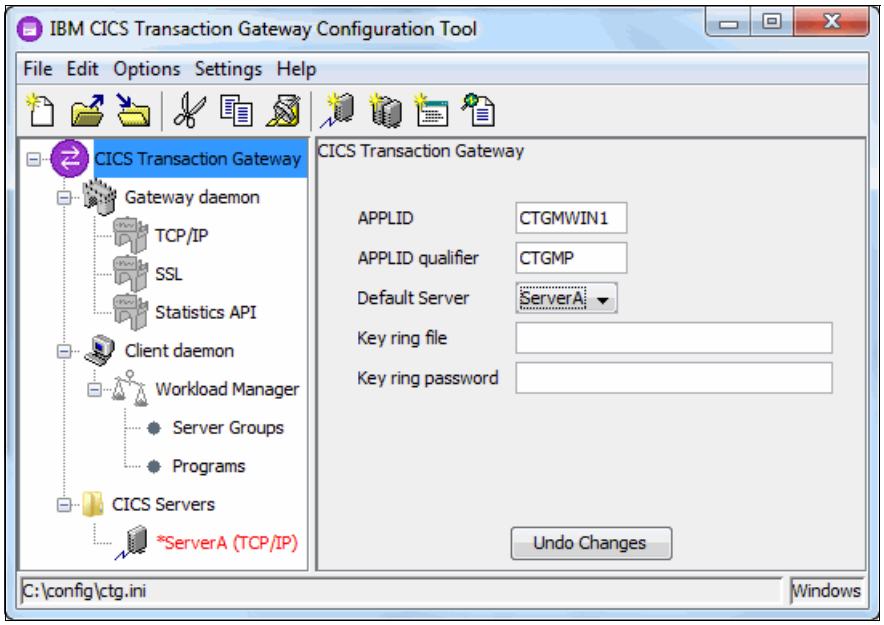


Figure 11-2 The CICS Transaction Gateway panel in the configuration tool

Table 11-1 summarizes the values used in the PRODUCT section of the CICS TG configuration file. The APPLID qualifier and APPLID values are not mandatory for the CICS TG for Multiplatforms or CICS TG Desktop Edition products. However, defining an APPLID qualifier and APPLID is a preferred practice for users of IPIC connections, because it helps to identify a Gateway daemon as an IPIC partner for administration, monitoring, and audit purposes. If they are omitted, CICS TG generates values specifically for IPIC connectivity.

The default server attribute is optional in the CICS TG configuration file, but it also represents a preferred practice. It is applied to application requests that do not specify a CICS server. The configuration tool forces the selection of a defined CICS server. However, at this stage, we have not defined our CICS servers, and so the only possible option is the example CICS server definition, ServerA.

Note: Some clients prefer that application requests that omit a value for the CICS server fail, rather than be routed to a default CICS server. This can be achieved by setting the default server to an undefined CICS server name. With the CICS TG for Multiplatforms and CICS TG for Desktop Edition products, this results in such application requests receiving an error, such as ECI_ERR_UNKNOWN_SERVER. This can only be achieved by manually editing the configuration file.

The default server attribute can be set to our intended value after the CICS server is defined, as explained in 11.2.5, “The IPICSERVER section” on page 216. The key ring attributes are used for Secure Sockets Layer (SSL) connectivity, and they are described in “Configure the Gateway daemon to use the new keystore” on page 234.

For more details, see the CICS Transaction Gateway for Multiplatforms Information Center:

http://pic.dhe.ibm.com/infocenter/cicstgmp/v9r0/index.jsp

Table 11-1 PRODUCT section user-initialized parameters

Configuration parameter	Parameter value	Parameter usage
Applid	CTGMWIN1	APPLID identifies the instance of CICS TG on server connections, in messages and data output, and tasks in a CICSplex.
Applid Qualifier	CTGMP	APPLID qualifier is a high-level qualifier for the APPLID.
Default Server	ServerA *(temporary)*	Specify a CICS server name that is used by CICS Transaction Gateway for client application requests in which no CICS server name is specified.

Figure 11-3 shows the PRODUCT section of the CICS TG configuration file, as it is generated by the configuration tool at this intermediate stage. If you attempt to save the configuration at this point, the configuration tool will alert you that there are errors, because the definition of ServerA is incomplete.

```
SECTION PRODUCT
    APPLID=CTGMWIN1
    APPLIDQUALIFIER=CTGMP
    DEFAULTSERVER=ServerA
ENDSECTION
```

Figure 11-3 Intermediate PRODUCT section generated from the configuration tool

11.2.4 The GATEWAY section

The GATEWAY section of the configuration file defines operating parameters for the Gateway daemon. In the CICS TG configuration tool, the definitions that are used to generate the GATEWAY section of the configuration file are defined using the Gateway daemon panel and its Resources, Logging, and Monitoring tabs, which include panels for each type of protocol handler.

Gateway daemon settings

Figure 11-4 shows the Resources tab on the Gateway daemon settings panel of the configuration tool. Default values are used for each attribute on this tab except for “Enable reading input from console.” This option is only suitable for development environments on UNIX or Linux platforms, where the Gateway daemon is invoked explicitly through the **ctgstart** command rather than with the CICS TG daemon script, **ctgd**, as described in 11.3.2, “The UNIX and Linux platforms” on page 224.

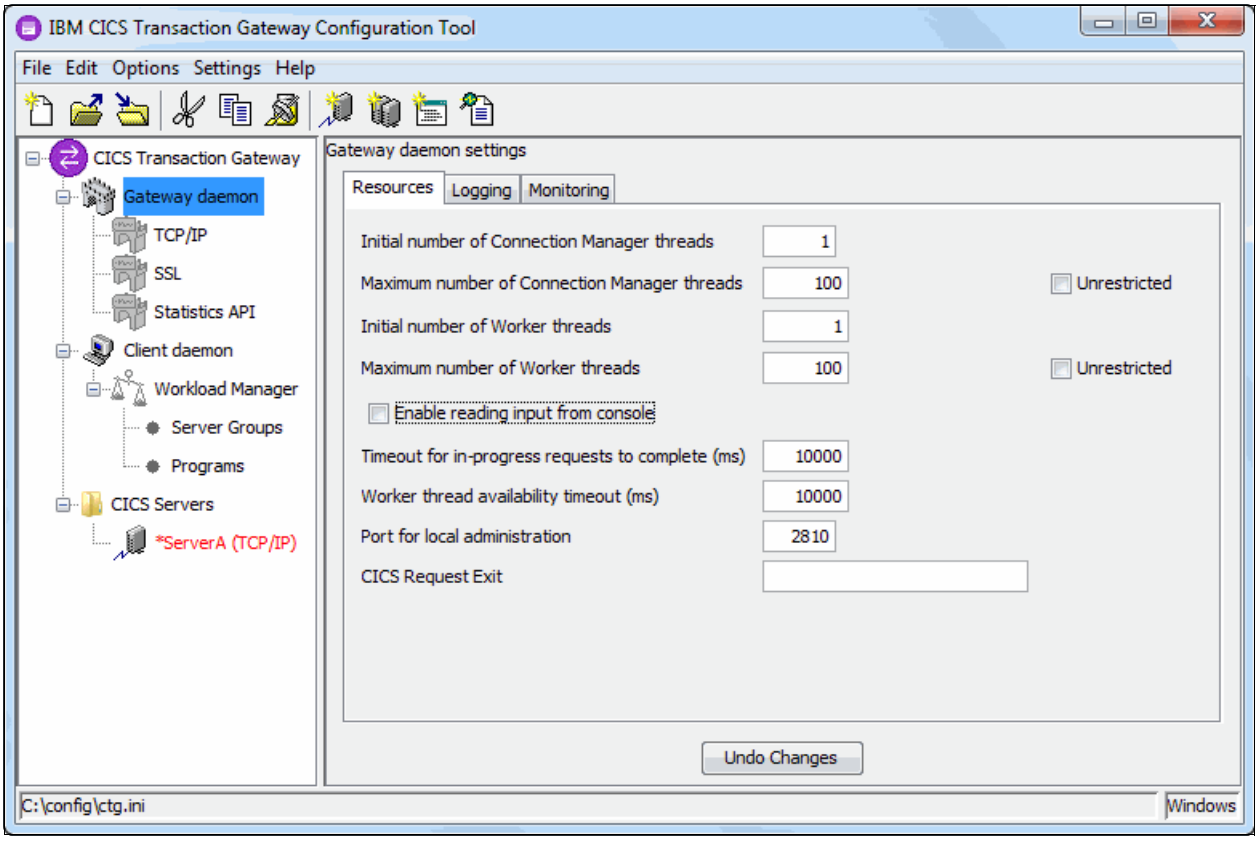


Figure 11-4 The Resources tab of the Gateway daemon settings panel of the configuration tool

Note: For the CICS TG Desktop Edition product, the following restrictions apply:

- Maximum number of connection manager threads (maxconnect) is limited to five.
- Maximum number of worker threads (maxworker) is limited to five.

Figure 11-5 on page 212 shows the GATEWAY section of the CICS TG configuration file generated by the configuration tool.

```
SECTION GATEWAY
    closetimeout=10000
    connectionlogging=on
    cicslogging=on
    initconnect=1
    initworker=1
    maxconnect=100
    maxworker=100
    noinput=on
    dnsnames=off
    trace=off
    workertimeout=10000
    adminport=2810
ENDSECTION
```

Figure 11-5 The first part of the GATEWAY section, generated by the configuration tool

Logging, monitoring, and statistics

The remaining two tabs in the Gateway daemon settings panel relate to log files (Figure 11-6, the Logging tab), request monitoring and statistics (Figure 11-7 on page 214, the Monitoring tab).

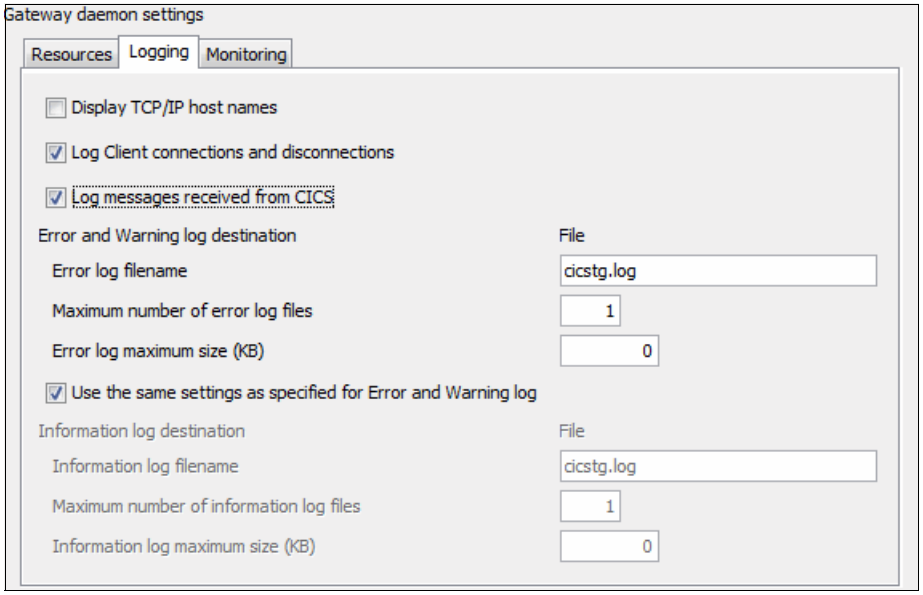


Figure 11-6 The Logging tab of the Gateway daemon settings panel

Gateway daemon log files

The option to display TCP/IP host names causes the Gateway daemon to perform a domain name server (DNS) lookup on client IP addresses, which can have a performance impact. This action can be useful in a development environment, but it is typically disabled for production systems.

Logging client connections provides a useful audit log of client application location and activity patterns, but it can produce excessive output if the remote applications do not cache their connections between requests.

Defining a single Gateway daemon log message file allows informational, warning, and error log messages to be written to a single destination. The choice of using a single log file or separate log files is up to you, but be aware that using a single file has the advantage of showing the error and warning messages within the context of informational log messages.

Location of Gateway daemon log files

Determine the location of the CICS TG log files to use a file name and a directory that are not part of the product installation directory structure. The following guidelines represent the preferred practice on each platform:

- Windows: The default location for the log files is the `IBM\CICS Transaction Gateway` subdirectory, in the Windows common application data directory, for example:

  `C:\ProgramData\IBM\CICS Transaction Gateway`

- UNIX/Linux systems: The default location for the log files is the `<install_path>/bin` directory. It is suggested to configure the Gateway daemon log files to use the default location of the Client daemon log files (`/var/cicscli`). This configuration can be implemented using the following pair of log handler definitions:

  ```
  log@info.dest=file
  log@info.parameters=filename=/var/cicscli/cicstg.log;maxfiles=1;filesize=0;
  log@error.dest=file
  log@error.parameters=filename=/var/cicscli/cicstg.log;maxfiles=1;filesize=0;
  ```

Configuring CICS TG statistics

The Monitoring tab is used to configure the timing of CICS TG statistics collections, and whether they are recorded to an external file at collection interval boundaries. Figure 11-7 on page 214 shows this configuration, where the default interval of 3 hours is accepted with a logical end of day at local midnight. (The regular statistics intervals align with the logical end of day.) Figure 11-7 on page 214 also shows that statistics recording is enabled. A full set of CICS TG statistics is encoded as XML at the end of each statistics interval and appended to the defined statistics log file, providing a historical record of CICS TG activity.

The specified `Statistics log filename` attribute is actually used as a template. Each time that the Gateway daemon starts, a time stamp is used to create a new statistics log file. The time stamp is inserted before the file name extension, or appended to the file name if no extension is specified. For example, using the configuration shown in Figure 11-7 on page 214, a statistics log file can be created when the Gateway daemon starts on this path:

`C:\monitoring\ctgstats_20140129-144745.xml`

If the Statistics log file name is defined to a location that is not available, the Gateway daemon issues an error log message, `CTG8476E`. Example 11-1 shows the error that is generated when the target directory (`C:\monitoring`) does not exist.

Example 11-1 The Gateway daemon error log message when the statistics log destination is not available

```
CTG8478E Unable to create statistics log file C:\monitoring\ctgstats_20140207-102525.xml, error:
C:\monitoring\ctgstats_20140207-102525.xml (The system cannot find the path specified.)
```

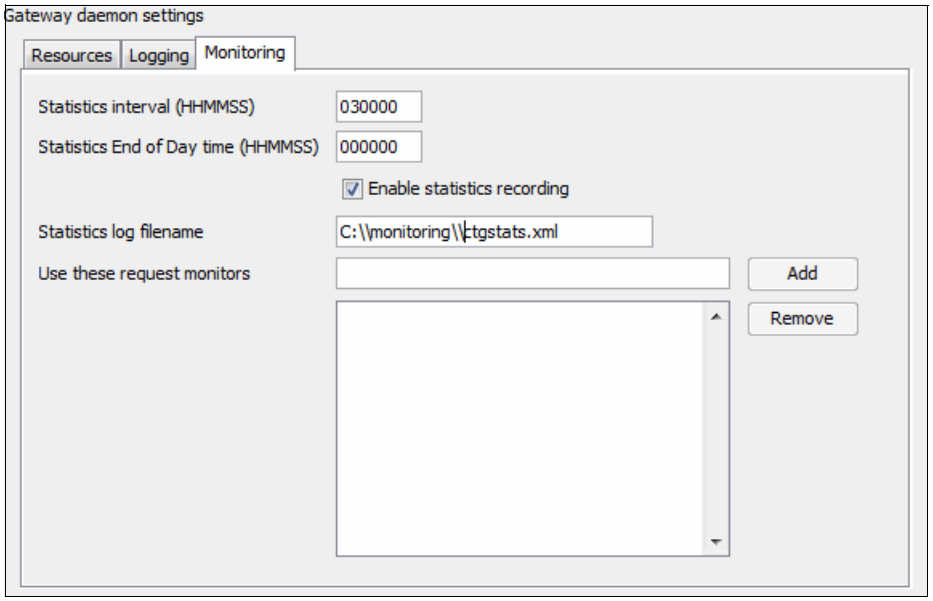


Figure 11-7 The Monitoring tab on the Gateway daemon settings panel of the configuration tool

Note: Statistics recording for CICS TG for Multiplatforms and CICS TG Desktop Edition is available in Version 9.0 or later.

A sample formatting program, `Ctgstat2`, is provided to process the XML-formatted statistics log file and produce human-readable output. For more details, search for the CICS TG for Multiplatforms Information Center topic, *Java Ctgstat2 statistics recording sample*:

http://ibm.co/1plHD3c

Request monitoring exits

Request monitoring exits are not defined in this configuration because monitoring and statistics are not the focus of this book. For details about configuring request monitoring exits, search for the CICS TG for Multiplatforms Information Center topic, *Request monitoring exits configuration*:

http://ibm.co/1szyold

Note: When the statistics log file is defined without a fully qualified path, the statistics log file is created relative to the default path for the platform. Here are the details:

- **Windows:** The log file is created in the `IBM\CICS Transaction Gateway` directory, in the Windows common application data directory, for example, `C:\ProgramData\IBM\CICS Transaction Gateway`. Note that double backslash characters (`\\`) are required to represent a single backslash in a Windows path due to Java string-formatting conventions.
- **UNIX and Linux:** The log file is created in the `/var/cicscli` directory.

These parameter settings cause the entries shown in Figure 11-8 on page 215 to be generated in the `GATEWAY` section of the configuration file.

```
log@statistics.dest=file
log@statistics.parameters=filename=C:\\monitoring\\ctgstats.xml

log@info.dest=file
log@info.parameters=filename=cicstg.log;maxfiles=1;filesize=0;

log@error.dest=file
log@error.parameters=filename=cicstg.log;maxfiles=1;filesize=0;

statint=030000
stateod=000000
statsrecording=on
```

Figure 11-8 The logging and statistics components of the configuration file

Configure the tcp and statsapi protocol handlers

The final part of the `GATEWAY` section contains the Gateway daemon protocol handler definitions. The protocol handlers define the entry points to the Gateway daemon for application requests, statistics API tools (such as the *CICS Explorer* and *IBM Tivoli OMEGAMON for CICS*), and administration functions through the **`ctgadmin`** utility.

A common option of the TCP, SSL, and Statistics API protocol handler definitions is the bind address. This attribute restricts the protocol handler to listening on a specific IP address, and it supports both IPV4 and IPV6 formats. The default for this parameter is no bind restrictions, and the protocol handler binds to all host addresses available on the system.

Figure 11-9 shows the TCP/IP settings panel in the CICS TG configuration tool that is used to define the `tcp` protocol handler.

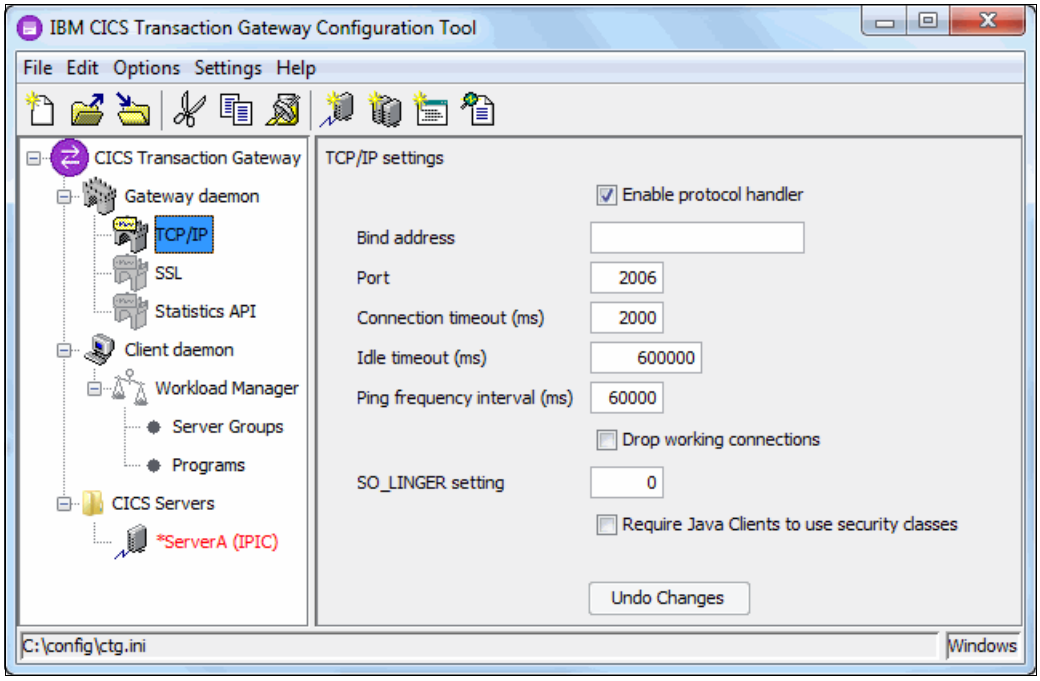


Figure 11-9 Gateway daemon TCP/IP settings panel of the configuration tool for the tcp protocol handler

Figure 11-10 shows the Gateway daemon `tcp` and `statsapi` protocol handler definitions that are generated by the configuration tool (the long second line wraps to a new line due to the limited width of this page).

```
protocol@tcp.handler=com.ibm.ctg.server.TCPHandler
protocol@tcp.parameters=connecttimeout=2000;idletimeout=600000;pingfrequency=60
000;port=2006;bind=;solinger=0;
protocol@statsapi.handler=com.ibm.ctg.server.RestrictedTCPHandler
protocol@statsapi.parameters=port=2980;bind=;connecttimeout=2000;maxconn=5
```

Figure 11-10 The tcp and statsapi protocol handler definitions generated by the configuration tool

Protocol handler definitions can be split between multiple lines using the backslash character as a line continuation marker. For examples, see the product sample configuration file, `ctgsamp.ini`.

Note: For this configuration, the default values for all attributes on the `tcp` and `statsapi` protocol handlers are used.

For more details about Gateway daemon `tcp` protocol handler definitions, search for the CICS TG for Multiplatforms Information Center topic, *TCP protocol parameters*:

http://ibm.co/1szyvNy

For more information about Gateway daemon `statsapi` protocol handler definitions, search for the information center topic, *Statistics API protocol parameters*:

http://ibm.co/1szyBF3

11.2.5 The IPICSERVER section

The `IPICSERVER` section of the configuration file defines a connection from the Gateway daemon to a CICS server using the IPIC protocol. The configuration tool creates `IPICSERVER` definitions in the configuration file whenever a CICS server is defined with the Network protocol attribute set to `IPIC`. If this attribute is set to `TCP/IP` or `SNA`, a SERVER definition is created.

As described in 11.2.3, “The PRODUCT section” on page 209, at least one CICS server, `ServerA`, is defined automatically, but incompletely, by the configuration tool. Figure 11-11 on page 217 shows this CICS server entry after it is modified to match the `TCPIPService` definition installed in `CICSTOR1` (as described in 9.2.3, “TCPIPSERVICE resource definition” on page 151) and the `IPConn` definition (described in “Defining the IPCONN for the Gateway daemon on Multiplatforms” on page 155).

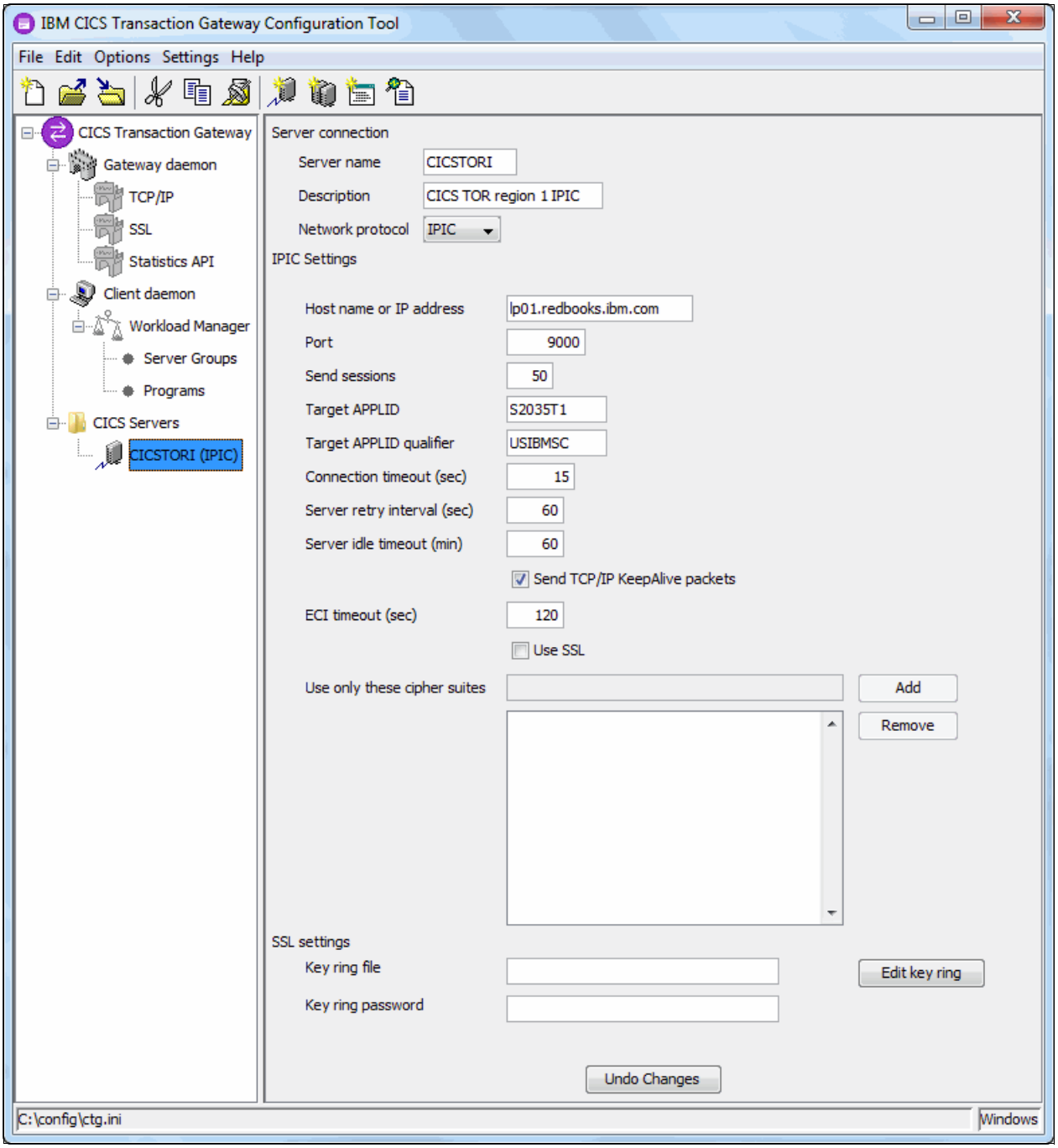


Figure 11-11 The Server connection panel of the configuration tool for IPIC connection CICSTORI

The Host name, Port, Send sessions, Target APPLID, and Target APPLID qualifier attributes are set to match the definitions in Chapter 9, “Configuring IPIC in CICS Transaction Gateway for z/OS” on page 147. Default values are acceptable for the Server retry interval, Send TCP/IP KeepAlive packets, and Use SSL attributes.

The Connection timeout default value of 0 means that the Gateway daemon will wait forever, or until there is a network error, to establish an IPIC connection to a CICS server, for example, if the server is not immediately available because it is being recycled. It is better to specify a non-zero value, but the correct value will depend on specific requirements.

If a CICS server is unavailable, the Server retry interval value added to the Connect timeout value represents the maximum time that an application might wait before receiving an error response. If a connection attempt does not complete before reaching the Connect timeout value, the Gateway daemon will not attempt to connect again until the Server retry interval expires. Application requests that are received by the Gateway daemon during the Server retry interval for the IPIC connection will immediately receive an error response.

Note: The `Server retry interval` attribute is available in CICS TG products starting with Version 8.1.

The Server idle timeout default value of `0` means that an established IPIC connection will remain connected when the workload stops. Again, there is no correct setting to suit all situations, but we chose a value of `60` (1 hour) simply to illustrate the decision point.

The ECI timeout attribute allows the system administrator to override application-specific timeout values. It is especially useful for situations when a badly designed application specifies an ECI timeout of `0`.

Note: The `ECITIMEOUT` attribute is available in CICS TG products starting with Version 9.0.

Figure 11-12 shows the `IPICSERVER` section of the CICS TG configuration file, which is generated by the configuration tool.

```
SECTION IPICSERVER = CICSTORI
    DESCRIPTION=CICS TOR region 1 IPIC
    SRVIDLETIMEOUT=60
    HOSTNAME=lp01.redbooks.ibm.com
    PORT=9000
    CONNECTTIMEOUT=15
    SRVRETRYINTERVAL=60
    TCPKEEPALIVE=Y
    SENDSESSIONS=50
    ECITIMEOUT=120
    CICSAPPLID=S20351T1
    CICSAPPLIDQUALIFIER=USIBMSC
    SSL=N
ENDSECTION
```

Figure 11-12 The IPICSERVER section for CICSTORI generated by the configuration tool

Note: For CICS TG Desktop Edition, IPIC send sessions (`sendsessions`) are limited to a maximum value of `5`.

Table 11-2 summarizes the attributes and values used to define the CICS server `CICSTORI`, representing an IPIC connection to the CICS region, `CICSTOR1`.

Table 11-2 IPICSERVER definition for CICSTORI

Configuration parameter	Parameter value	Parameter usage
Server name	`CICSTORI`	Use this name for all requests to access the server from client applications.
Description	`CICS TOR region 1 IPIC`	The description for the server. This is optional but useful to identify this server definition.
Hostname	`lp01.redbooks.ibm.com`	The host name to which the CICS `TCPIPService` is bound.
Port	`9000`	The port number on which the target CICS server is listening.
Send sessions	`50`	The number of simultaneous transactions, or CICS tasks, that are allowed over the connection. This is limited by the `ReceiveCount` attribute of the CICS `IPCONN` definition.
Target APPLID	`S20351T1`	The APPLID of the target CICS server. This field is optional unless the CICSAPPLIDQUALIFIER is specified. If specified, the CICSAPPLID must match the APPLID of the target CICS server.
Target APPLID qualifier	`USIBMSC`	This field is optional unless CICSAPPLID is specified. If specified, the CICSAPPLIDQUALIFIER must match the network ID of the target CICS server.
Use SSL	`N`	This indicates that the connection is not using the SSL protocol.
Use only these cipher suites	Required ciphers or blank	This is only required for SSL connections.

For additional information about defining IPIC connections, see the *Configuring IPIC* topic in the CICS Transaction Gateway for Multiplatforms Information Center:

http://ibm.co/1qPtXid

Update the default server definition

With the CICS server `CICSTORI` now defined, it can now be selected as the default server on the CICS Transaction Gateway panel of the configuration tool. Figure 11-13 shows the updated `PRODUCT` section of the CICS TG configuration file.

```
SECTION PRODUCT
    APPLID=CTGMWIN1
    APPLIDQUALIFIER=CTGMP
    DEFAULTSERVER=CICSTORI
ENDSECTION
```

Figure 11-13 Completed PRODUCT section including the correct default server definition

The CICS TG configuration file can now be used to start the Gateway daemon and run a test Java application.

11.3 Starting and stopping CICS TG

Depending on the platform that you are using, you can start and stop CICS TG using several methods. This section explains how to use Services for Windows platforms and the **ctgd** command for UNIX platforms. It also describes using the **ctgadmin** command on all distributed platforms. For details about these topics, see the CICS Transaction Gateway Information Center:

http://pic.dhe.ibm.com/infocenter/cicstgmp/v9r0/index.jsp

> **Note:** The full name of the Windows service installed by the CICS TG products is IBM CICS Transaction Gateway. In this chapter, it is referred to as the CICS TG service.

11.3.1 The Windows platform

For the Windows platform, the suggested method of starting CICS TG in a production environment is by using the Services option. This allows the CICS TG service to start automatically when the Windows machine is restarted.

The **ctgadmin** command provides an alternative means of controlling the CICS TG service, more suited to development and test environments, where configuration options are likely to be changed more frequently. For more detail, see “Using the ctgadmin command” on page 223.

When using the Windows service, remember the following important tips:

- To interact with Services through the standard Windows services administration interfaces, you must have Administrator authority.
- You can interact with Services using either the Local Server Services manager or the command line. Here is an example of starting the CICS TG service from the command line:

  ```
  net start "IBM CICS Transaction Gateway"
  ```

  > **Note:** CICS TG for Multiplatforms V9.0 and CICS TG Desktop Edition V9.0 APAR PI06582 allow an Administrator to grant permission for starting and stopping the CICS TG service to any user or group.

- After the CICS TG configuration file is in place and verified, it makes sense to modify the Startup Type to Automatic to ensure that CICS TG is always initialized following events, such as a reboot of the Windows machine. Follow these steps:

  a. From the Control Panel, choose **Administrative tools** to go into the Services administration window (Figure 11-14 on page 221) on the system that is running CICS TG.

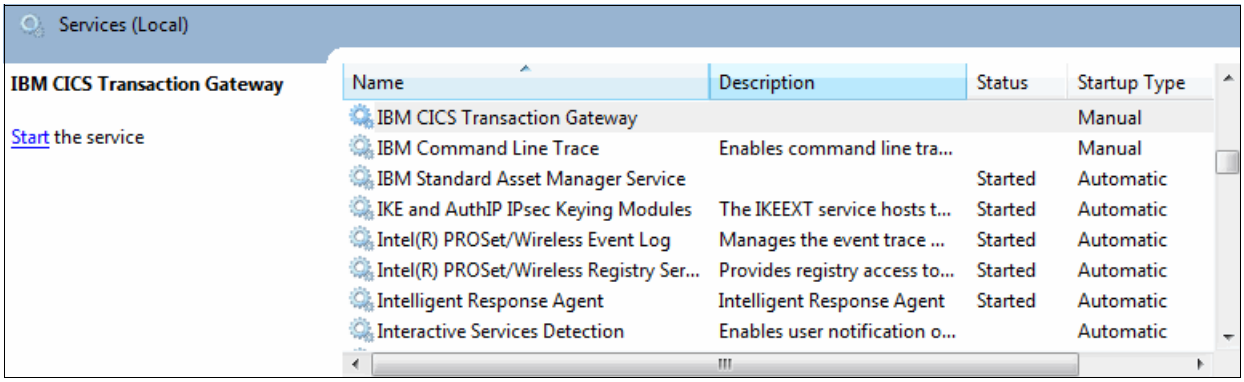


Figure 11-14 The IBM CICS Transaction Gateway entry in the Services administration window

b. Search through the services on the system until you find the IBM CICS Transaction Gateway service. By default, the Startup Type is set to Manual. Right-click the service entry and select **Properties**.

c. In the dialog box labeled IBM CICS Transaction Gateway Properties (Local Computer), go to the **General** tab and select **Automatic** from the Startup Type drop-down list, as shown in Figure 11-15.

d. Click **Apply**.

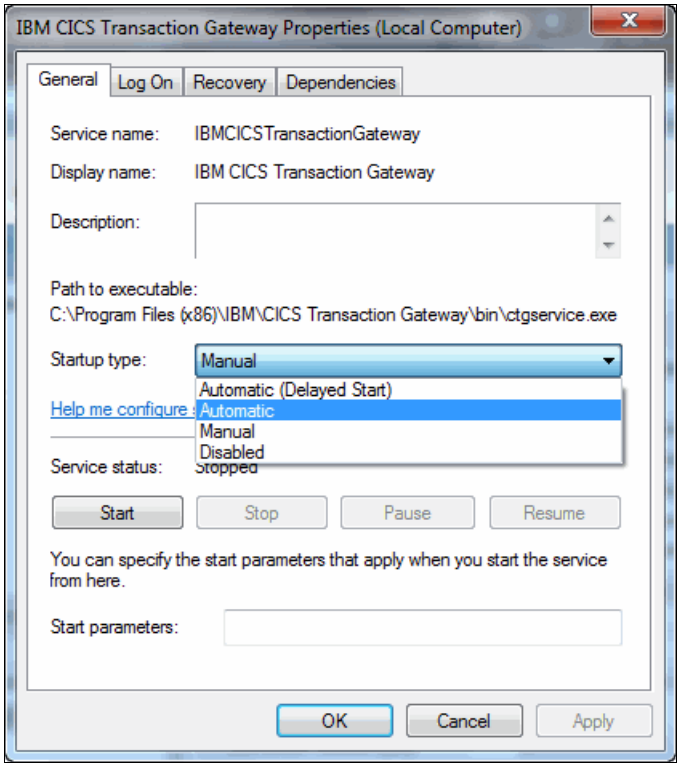


Figure 11-15 Modifying the startup type for the IBM CICS Transaction Gateway service

With the Startup type set to `Automatic`, the CICS TG service will now start whenever the system is restarted.

- You can still choose to control the service manually, such as for a development or test system where the configuration is modified frequently. Simply click Start or Stop in the Service administration window (Figure 11-14).

Note: If the CICS TG service fails to start, a generic Windows error dialog box suggests that you “Review the System Event Log and Contact the service vendor”. It can also provide a service-specific error code, such as `109` even though “109” is not a documented CICS TG error code. Contrary to the advice in the Windows error dialog box, better guidance for such problems is provided in the Application Event Log. Messages, such as `CTG9549E`, refer to appropriate further diagnostics, such as the Gateway daemon error log.

Overriding the CICS TG configuration

Certain characteristics of the Gateway daemon cannot be determined by attributes specified in the CICS TG configuration file. One example is a parameter specifically for the Java virtual machine (JVM), such as the maximum heap size. The size of the Java heap must sometimes be increased from the default size to support larger workloads.

Note: On Multiplatforms, the default Java heap size is `128` MB.

It can also be useful to make adjustments in a development environment, such as to evaluate the effects of a change, without modifying the CICS TG configuration file, or to override default values in a shared CICS TG configuration file.

The **`ctgservice`** command on Windows allows you to make these customizations. Overrides that are specified using the **`ctgservice`** command are stored in the Windows registry and remain in place when the CICS TG service is restarted or the machine is restarted.
Figure 11-16 shows the output from using the **`ctgservice`** command shown here to specify a 512 MB Java heap and a request monitoring exit for the Gateway daemon:

```
ctgservice -R -A-j-Xmx512
    -A-requestExits=com.ibm.ctg.samples.requestexit.BasicMonitor
```

```
CTG9512I CTGSERVICE - CICS Transaction Gateway Windows Service Utility
CTG9513I (C) Copyright IBM Corporation 2009, 2013.  All rights reserved.

CTG9518I Gateway daemon parameter overrides are:
   -A-j-Xmx512
   -A-requestExits=com.ibm.ctg.samples.requestexit.BasicMonitor
```

Figure 11-16 Using the ctgservice command to configure the Gateway daemon

Figure 11-17 on page 223 shows the help text from the **`ctgservice`** command.

```
C:\Users\Administrator>ctgservice -?
CTG9512I CTGSERVICE - CICS Transaction Gateway Windows Service Utility
CTG9513I (C) Copyright IBM Corporation 2009, 2013.  All rights reserved.

Command options are:
-R  Register Gateway daemon parameter overrides
    [-A<Param1> [-A<Param2> ...]]
-T  Display Gateway daemon parameter overrides

The following parameter overrides are valid:
  -port=<port_number>      - TCP/IP port for tcp protocol
  -sslport=<port_number>   - SSL port for ssl protocol
  -keyring=<keyring>       - keyring for use with the ssl protocol
  -keyringpw=<keyring_pw>  - keyring password for use with the ssl protocol
  -adminport=<port_number> - TCP/IP port for local administration
  -statsport=<port_number> - TCP/IP port for statistics API requests
  -initconnect=<number>    - Initial number of connection manager threads
  -maxconnect=<number>     - Maximum number of connection manager threads
  -initworker=<number>     - Initial number of worker threads
  -maxworker=<number>      - Maximum number of worker threads
  -trace                   - Enable standard trace
  -dnsnames                - Use DNS to display symbolic TCP/IP host names
  -tfile=<filename>        - Trace file name
  -x                       - Enable full detailed trace
  -tfilesize=<number>      - Maximum trace file size in kilobytes
  -truncationsize=<number> - Maximum size of trace data blocks in bytes
  -dumpoffset=<number>     - Byte offset in data to start trace output
  -stack                   - Switch on exception stack trace
  -j<argument>             - argument to pass to the JVM
  -classpath=<class path>  - additional entries to append to JVM class path
  -requestExits=<exits>    - list of classes to be used for request exit
monitoring

Parameter overrides override those in the configuration file
```

Figure 11-17 Command-line help output for the ctgservice command

Using the ctgadmin command

You can also interact with Windows services by using the **ctgadmin** command. To begin, type this string:

```
C:\Users\Administrator>ctgadmin -a start
```

The details are listed for the full suite of **ctgadmin** command options for the Windows platform. Note that both a **start** option and a **shut** option are available. For more information about action-specific options, enter `-?` after specifying `-a` or `-msg` (for example, **ctgadmin -a trace -?**).

- Usage:

  ctgadmin [**-adminport** *<number>*] [**-dbg** [*<filename>*]] **-a** *<action>* [*<action specific options>*] | **-msg** *<message id>*

- Command options:

-a *<action>*	The action to be performed
-adminport *<number>*	The port used to communicate with the CICS TG service (default is 2810)
-dbg [*<filename>*]	Output trace to stderr, or *<filename>* if specified
-msg *<message id>*	Display an explanation for the specified message

- Available actions:

`start`	Start the CICS TG service
`shut`	Shut down the CICS TG service
`trace`	Display or change Gateway daemon trace settings
`stats`	Display statistics about the CICS TG
`dump`	Requests status dumps from the CICS TG
`rmexit`	Interact with the request monitoring exits
`crexit`	Interact with the CICS request exit

11.3.2 The UNIX and Linux platforms

For the UNIX and Linux platforms, while you can manually control the Client and Gateway daemon processes independently through the command-line shell, it is a preferred practice to control both daemon processes as a pair. Operating both daemon processes as a single service is a preferred practice for production systems, because restarting one without the other is not a supported operation.

The supplied daemon script, **ctgd**, provides a UNIX System V service interface where you can enable both Client and Gateway daemon processes to start during system initialization.

> **Note:** The **ctgd** command provides a way for both the system itself, and a system operator, to control the CICS TG service.

Independent control of the Client and Gateway daemon processes can be appropriate for a development or test system, where the configuration is modified frequently. In these instances, the **ctgstart** command is used directly from the command-line shell to start the Gateway daemon. With command-line input enabled, the Gateway daemon can be shut down by entering `Q` (quit) or `I` (Immediate) in the **ctgstart** shell session. Similarly, the **cicscli** command can be used to manually administer the Client daemon.

Configuring CICS TG as a daemon process

To configure the CICS TG daemon process by using **ctgd**, the system administrator must customize a copy of the sample shell script, **ctgdsamp.conf**. This secondary shell script is invoked by **ctgd** to establish several necessary details about the system through specific environment variables:

- The install path to Java
- The location of the CICS TG configuration file
- Configuration overrides for the **ctgstart** script
- The Gateway daemon administration port (used by the **ctgadmin** command)

  This overrides the port number specified in the `restrictedtcphandler`, defined in the CICS TG configuration file. This is useful when using a shared configuration file.

The sample configuration script for **ctgd**, `ctgdsamp.conf`, includes comments for each environment variable. It is provided in the product directory:

`<install_path>/samples/configuration`

Example 11-2 shows a sample configuration script for **ctgd**, including override values for the Gateway daemon in the **CTGD_PARAMS** environment variable (where the Gateway daemon JVM is configured with a maximum Java heap size of 512 MB and a request monitoring exit). The **ctgstart** script requires the user and group security attributes for the Gateway daemon and Client daemon processes, plus the port number where the Gateway daemon will listen for ctgadmin requests. The three exported variables shown in the sample, *CICSCLI*, *CTG_JAVA*, and *CLASSPATH*, are explicitly exported because they are required by the CISC TG processes at run time. Other script variables with a prefix of *CTGD* are used exclusively within the scope of the **ctgd** daemon script and therefore do not require an export statement.

Example 11-2 Sample ctgd configuration script that is called by ctgd and defined by CTGDCONF

```
CTGD_USER=nobody
CTGD_GROUP=nobody
CTGD_ADMINPORT=2810
CTGD_PARAMS="-j-Xmx512M -requestExits=com.ibm.ctg.samples.requestexit.BasicMonitor
"
export CICSCLI=/opt/ibm/cicstg/bin/ctg.ini
export CTG_JAVA=/opt/ibm/cicstg/jvm17/bin/java
export CLASSPATH=/opt/ibm/cicstg/classes/ctgsamples.jar
```

Note: Values specified in the **ctgd** configuration script override equivalent attributes specified in the CICS TG configuration file.

The daemon script, **ctgd**, uses the environment variable **CTGDCONF** to determine the configuration script's location and file name. This means that **CTGDCONF** must be set at the system level for **ctgd** to locate the configuration script during system initialization. UNIX and Linux system administrators can use their preferred method of setting **CTGDCONF**, if it is available when **ctgd** is invoked during system initialization.

If the **CTGDCONF** environment variable is not set, **ctgd** expects to find a script in the default location:

`<install_path>/bin/ctgd.conf`

Using the ctgd script

After it is configured, **ctgd** can be used to start the CICS TG service automatically during system initialization and stop the CICS TG service automatically during system shutdown. In this way, **ctgd** manages both the Gateway daemon and Client daemon processes in unison.

Automatic control of the CICS TG daemon service can be implemented according to the convention best suited to a particular operating system. Systems administrators on AIX might prefer to add an entry for **ctgd** to the file `/etc/inittab`, for example:

`<install_path>/bin/ctgd`

Alternative implementations rely on adding a symbolic link to the **ctgd** shell script, `<install_path>/bin/ctgd`, to the appropriate system directory, for example, `/etc/init.d` on Linux systems.

The **ctgd** script can also be used directly by a system administrator to start and stop the CICS TG service. To run the **ctgd** script from a command line, open a shell session and enter the following command:

ctgd {start|stop}

> **Remember:** You must have a valid `ctgd.conf` file to execute the **ctgd** command. The default location and name for the ctgd configuration file is <*install path*>/bin/ctgd.conf.

For more information about **ctgd**, search for the CICS TG for Multiplatforms Information Center topic, *Running the Gateway daemon as a background process*:

http://ibm.co/1mphTA7

Using the ctgadmin command

The **ctgadmin** command provides the ability to interact with the Gateway daemon and has one overlapping feature with **ctgd**, which is shutdown. The preferred practice is to use **ctgd** for start or stop operations, and **ctgadmin** for other administrative purposes.

A full list of **ctgadmin** command functions on UNIX and Linux follows:

- Usage:

  ctgadmin [**-adminport** <*number*>] [**-dbg** [<*filename*>]] **-a** <*action*> [<*action specific options*>] | **-msg** <*message id*>

- Command options:

-a <*action*>	The action to be performed
-adminport <*number*>	The port used to communicate with the Gateway daemon (default is 2810)
-dbg [<*filename*>]	Output trace to the standard error destination (`stderr`) stderr, or <*filename*> if specified
-msg <*message id*>	Display an explanation for the specified message

- Available actions:

`shut`	Shut down the Gateway daemon
`trace`	Display or change Gateway daemon trace settings
`stats`	Display statistics about the CICS TG
`dump`	Requests status dumps from the CICS TG
`rmexit`	Interact with the request monitoring exits
`crexit`	Interact with the CICS request exit

> **Note:** In contrast to **ctgadmin** on Windows operating systems, there is no `start` action on UNIX and Linux systems because running the CICS TG processes as a service is not mandatory. If **ctgd** is not configured, there can be no daemon service to start.

Using console mode

The final way to administer CICS TG is by using the **ctgstart** script, and the **cicscli** command, through a user console. This method is acceptable for a development or test system where the configuration is modified frequently. Be aware that the CICS TG processes run under the security context of the console user.

The **ctgstart** script is used to start both the Gateway daemon and the Client daemon. It has this syntax:

ctgstart <*options*> [<-*j*>JVMArg1 <-*j*>JVMArg2...] [<-*c*>CicscliArg1 <-*c*>CicscliArg2...]

The **ctgstart** script accepts many optional parameters, shown in Example 11-3 on page 227, to override attributes that are otherwise taken from the CICS TG configuration file.

Example 11-3 Parameters for the ctgstart script

```
-port=<port_number>        TCP/IP port for TCP protocol
-sslport=<port_number>     SSL port for SSL protocol
-keyring=<keyring>         Key ring for use with the SSL protocol
-keyringpw=<keyring_pw>    Key ring password for use with the SSL protocol
-adminport=<port_number>   TCP/IP port for local administration
-statsport=<port_number>   TCP/IP port for statistics API requests
-initconnect=<number>      Initial number of connection manager threads
-maxconnect=<number>       Maximum number of connection manager threads
-initworker=<number>       Initial number of worker threads
-maxworker=<number>        Maximum number of worker threads
-trace                     Enable standard trace
-quiet                     Disable reading/writing to/from console
-dnsnames                  Use DNS to display symbolic TCP/IP host names
-tfile=<filename>          Trace file name
-x                         Enable full detailed trace
-tfilesize=<number>        Maximum trace file size in kilobytes
-truncationsize=<number>   Maximum size of trace data blocks in bytes
-dumpoffset=<number>       Byte offset in data to start trace output
-stack                     Switch on exception stack trace
-j<argument>               Argument to pass to the JVM
-classpath=<class path>    Additional entries to append to JVM class path
-c<argument>               Argument to pass to the cicscli command
-requestExits=<exits>      List of classes to be used for request exit monitoring
```

Note: The Client daemon is administered through the **cicscli** command, but that command is not described here because the focus of this chapter is IPIC connectivity and security. The Client daemon is not used for IPIC connectivity.

11.4 Testing the Gateway daemon

You can now test the Gateway daemon that you configured in 11.2, “Configure the Gateway daemon” on page 207.

This section explains how to test the Gateway daemon by using the Java sample programs, `EciB1` and `EciB3`. The source code for these sample programs is in the product directory:

`<install_path>\samples\java\com\ibm\ctg\samples\eci`

There is no need to compile the Java samples from the source code because the CICS TG product installation includes a JAR file with precompiled versions of each sample. To use the precompiled Java samples, ensure that the CICS TG product files `ctgsamples.jar` and `ctgclient.jar` are included on the **`CLASSPATH`** environment variable of the test system.

11.4.1 Start the Gateway daemon

Verify that the Gateway daemon initialized successfully. Verify that it is listening for application requests on the intended port number. Example 11-4 on page 228 shows selected log messages from the Gateway daemon initialization.

Example 11-4 Log messages verifying Gateway daemon configuration and initialization

```
CTG8400I Using configuration file C:\config\ctg.ini
CTG8426I The APPLID is CTGMWIN1
CTG8427I The APPLID qualifier is CTGMP
CTG6577I Java details: Version=1.7.0 - 32-bit, Path=C:\Program Files
(x86)\IBM\CICS Transaction Gateway\jvm17\bin\java.exe
CTG6502I Initial Connection managers = 1, Maximum Connection managers = 100
CTG6526I Initial workers = 1, Maximum workers = 100
CTG6505I Successfully created the initial connection manager and Worker threads
CTG6524I Successfully started handler for the tcp: protocol on port 2006
CTG6524I Successfully started handler for the statsapi: protocol on port 2980
CTG8481I Statistics recording is enabled to statistics log file
C:\monitoring\ctgstats_20140207-103224.xml
CTG6512I CICS Transaction Gateway initialization complete
CTG6411I Interval statistics are active with the statistics interval length set to
3 hours, 0 minutes and 0 seconds
CTG6415I The statistics end of day time is set to 00:00:00 Greenwich Mean Time
```

11.4.2 Test the connection

Use the `EciB1` sample to run a basic test to prove connectivity with the Gateway daemon. You can run this sample in remote mode to flow an ECI request through the Gateway daemon to a CICS server. Provided the CICS server has the `EC01` program installed, as described in 9.3, “The sample application programs” on page 156, `EciB1` returns the current date and time, as determined by `EC01`.

This program is useful for testing connectivity to the Gateway daemon and CICS server, regardless of the platform on which CICS TG is installed or how the CICS server is connected. With the environment configured, run the following command:

```
Java com.ibm.ctg.samples.eci.EciB1 localhost 2006
```

> **Note:** If CICS TG is installed on a different machine than your test system, ensure that you use the IP address or host name for that machine rather than *`localhost`*, as shown in the example.

The sample program presents a list of CICS servers that are defined in the Gateway daemon configuration file and prompts you to select one of those servers. In the example shown in Figure 11-18 on page 229, the TCP/IP protocol server is selected and the time and date from the CICS server are returned.

```
CICS Transaction Gateway Basic ECI Sample 1

Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway URL]
                                          [Gateway port number]
                                          [SSL keyring]
                                          [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 localhost port:2006.

CICS servers defined:

         1. CICSTORI -CICS TOR region 1 IPIC


Choose server to connect to, or q to quit:
1

Program EC01 returned with data:-

        Hex: 32302f30362f31332030353a31323a34340
        ASCII text: 20/06/13 05:12:44
```

Figure 11-18 EciB1 test output using the IPIC connection

11.4.3 Test the channel and container support

Perform further testing with the `EciB3` sample program, which uses channels and containers. This program is supported only when using the IPIC protocol. `EciB3` requires that the CICS server program `EC03` is available, as described in 9.3, “The sample application programs” on page 156.

Run the following command:

```
Java com.ibm.ctg.samples.eci.EciB3 localhost 2006
```

If this test runs successfully, it sends user-entered text using channels and containers to the selected CICS server. The data that is sent to the `EC03` program is returned in an output container, together with separate containers for the current date and time, and the length of the user data. See Figure 11-19 on page 230.

```
CICS Transaction Gateway Basic ECI Sample 3

Usage: java com.ibm.ctg.samples.eci.EciB3 [Gateway URL]
                                          [Gateway port number]
                                          [SSL keyring
                                           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 localhost port 2006
CICS servers defined:
         1. CICSTORI -CICS TOR region 1 IPIC

Choose server to connect to, or Q to quit: 1

Enter text to send to the CICS program: This is REDBOOKS test data

Program ECO3 returned 4 containers in channel "SAMPLECHANNEL":
        [CHAR] CICSDATETIME     = 20/06/2013 07:34:50
        [CHAR] INPUTDATA        = This is REDBOOKS test data
        [CHAR] OUTPUTMESSAGE    = Input data was: This is REDBOOKS test data

         [BIT] INPUTDATALENGTH  = 0000001a
```

Figure 11-19 EciB3 test output using IPIC connection with ECI channel and container support

11.5 Configuring secure connections

In a remote mode configuration, SSL can be used on client connections to the Gateway daemon and on `IPIC` connections between the Gateway daemon and the CICS server (CICS TG for z/OS regions and TXSeries regions). This section describes the steps required to configure and test both parts of a secure 3-tier system.

The following levels of security are possible:

- Secure Sockets Layer (SSL) server authentication
- SSL client authentication

During server authentication, a connection is established only if the client trusts the server based on the information that is presented by the server to the client in its certificate. If client authentication is enabled, the client also sends its certificate information to the server. A connection is then established only if the client trusts the server and if the server trusts the client, based on the information exchanged in both certificates.

In addition, the following types of certificates can be used:

- Self-signed certificates
- Signed authority certificates

The *signed authority certificate* is typically provided to you (or purchased). The *self-signed certificate* is created by the user and it is considered less secure, but it is adequate for the purpose of this book. The following subsections use a self-signed client certificate for the application-to-Gateway daemon secure connection, and they use the exported self-signed certificate authority (CA) from the CICS configuration (described in 9.4.1, “Create a key ring for CICS” on page 163) for the Gateway-to-CICS secure connection.

Note: The creation of Java keystores in the following subsections is described using the IBM Key Management (**iKeyman**) tool. However, this tool is only available with an IBM-supplied Java Runtime Environment (JRE). The preferred practice today is to use the **keytool** command-line utility, which is included in IBM and non-IBM JRE implementations. You can use **keytool** in place of **iKeyman** to create the Java keystores described in this chapter, using the same syntax shown in “Create and populate the client keystore” on page 190.

11.5.1 Secure the Gateway daemon

The first step in establishing secure connectivity to or from the Gateway daemon is to provide it with the necessary credentials to authenticate itself with securely connected components. Here, a self-signed personal certificate is created for the Gateway daemon.

Create and populate the Gateway daemon key ring

To configure the Gateway daemon for SSL server authentication, first create a Java keystore and a certificate for the Gateway daemon to identify itself. To create a Java keystore and Gateway daemon personal certificate, use the **iKeyman** tool, which is in the `jvm/bin` directory of CICS TG product installation. To start the tool, enter **iKeyman** at a command prompt or through Windows Explorer, as shown in Figure 11-20. Alternatively, you can use the **keytool** command-line utility, which is in the same directory.

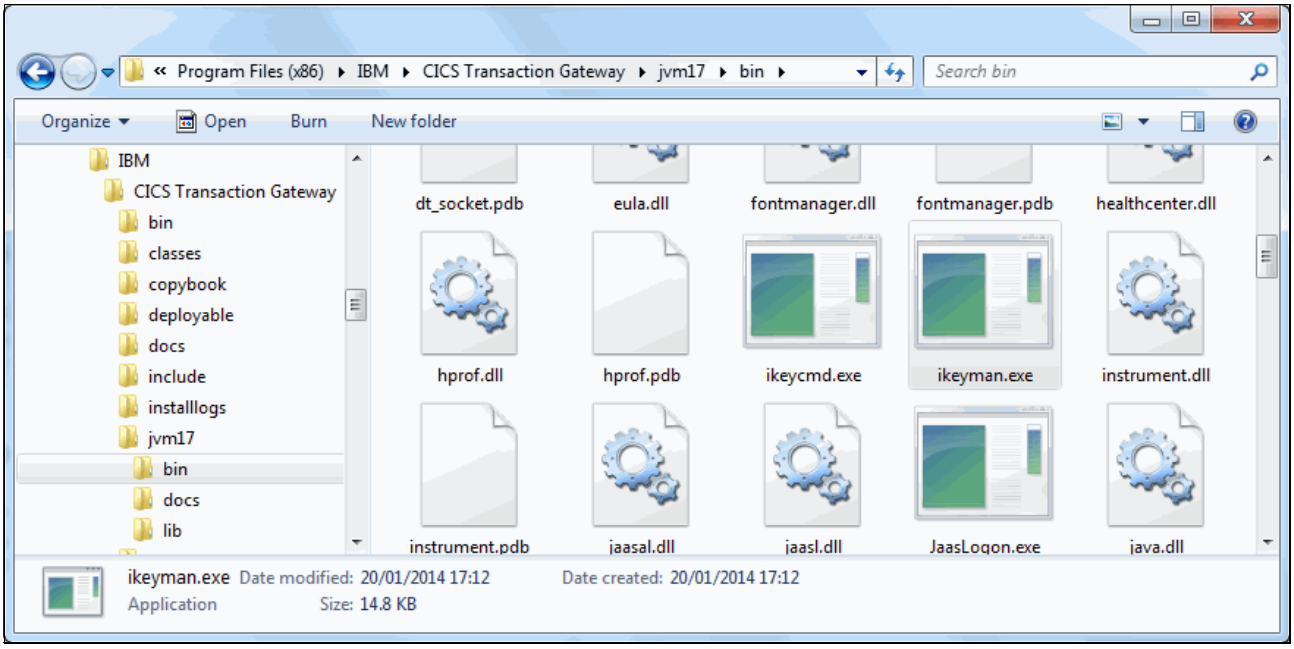


Figure 11-20 Locating and launching the iKeyman tool on Windows

Figure 11-21 shows the window that opens when the tools starts, before a keystore is created or opened.

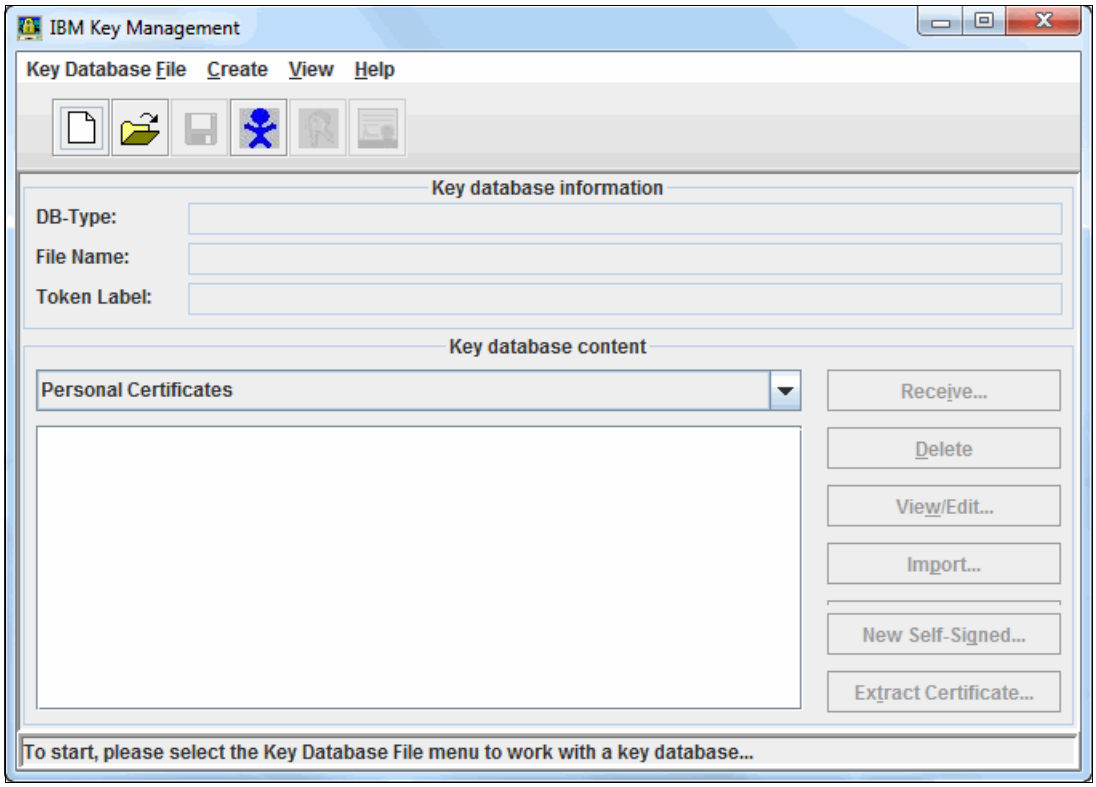


Figure 11-21 The opening view of the iKeyman tool

Follow these steps to create the Gateway daemon personal certificate by using the **iKeyman** tool:

1. Select **Key Database File** → **New** to create a Java keystore file.

   Set the Key database type to JKS and choose a suitable location and file name (in this example, we use `C:\config\Gateway-MWIN1.jks`). In addition, provide a password for the Gateway daemon Java keystore (we use the password `cmVkYm9vawZ9`).

   If the Java keystore is created, **ikeytool** returns to the default view and shows the following status message:

   ```
   The requested action has successfully completed! The current key store is also
   shown in ikeyman Window title.
   ```

2. Select **Personal Certificates** from the Key database content drop-down list, if it is not already selected, and then click **New Self-Signed**. This action displays the dialog box for creating a new self-signed certificate, shown in Figure 11-22 on page 233.

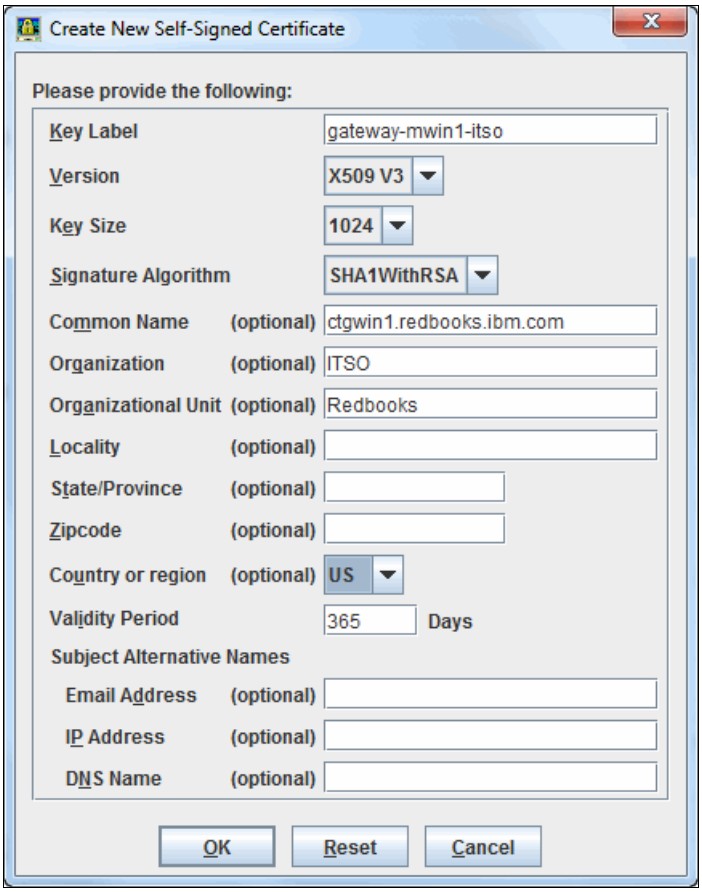


Figure 11-22 Create New Self-Signed Certificate dialog box

3. Complete the certificate request based on the guidelines in Table 11-3. The subset of entry fields used is adequate for the purpose of this book, but you can choose to include more details, as appropriate.

Table 11-3 Gateway daemon certificate values

iKeyman field	Value
Key Label	Enter a label to identify the Gateway certificate, such as `Gateway-mwin1-itso`.
Version	Select **X509 V3**.
Key Size	Select **1024**.
Common Name	Enter the host name of the Gateway daemon machine, such as `ctgwin1.redbooks.ibm.com`.
Organization	Enter the name of your organization, such as `ITSO`.
Organizational Unit	Enter the unit within your organization, such as `Redbooks`.
Country	Select a two-character ID from the list.
Validity Period	The default is `365` days.

4. Select **OK**. This selection generates a certificate that includes a public/private key pair and places them in the Java keystore. The name of the self-signed certificate, as you entered it in the Key Label field, is displayed in the Personal Certificates area of the window (Figure 11-23).

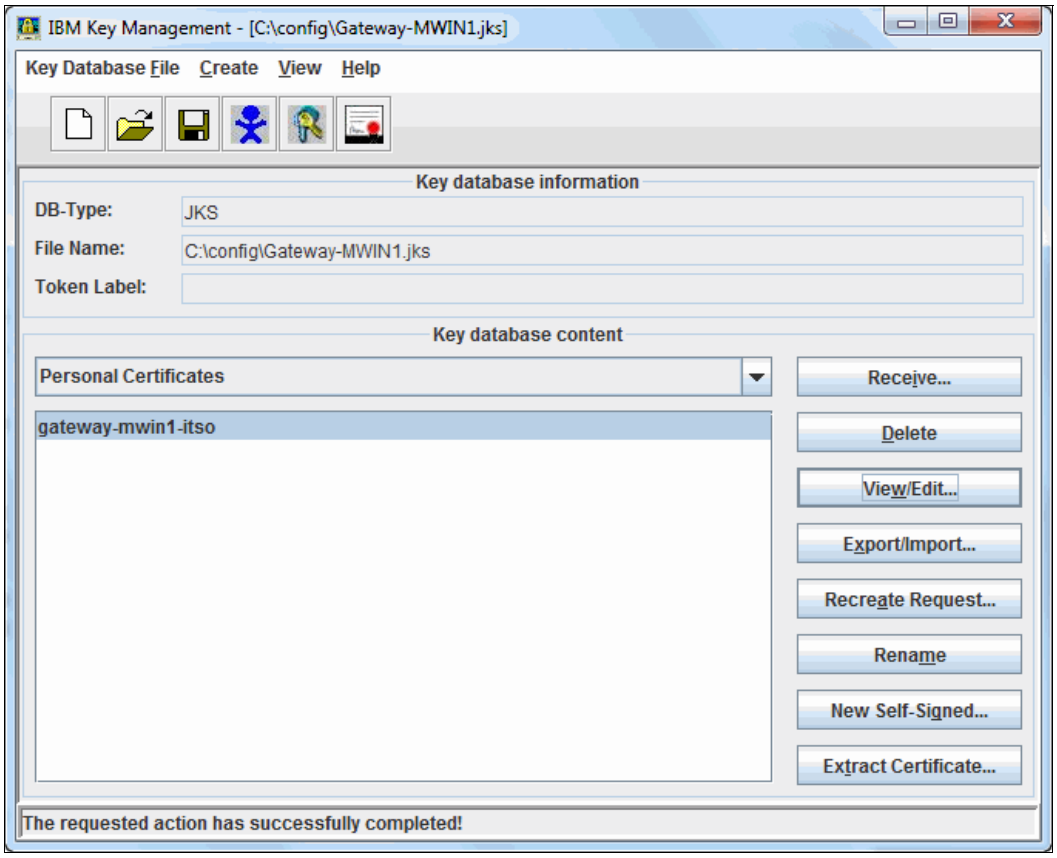


Figure 11-23 The Gateway daemon keystore with a self-signed personal certificate

Configure the Gateway daemon to use the new keystore

Now that the certificates are in place, you can configure CICS TG to use them. This involves updating the `PRODUCT` section of the CICS TG configuration file with the details of the Gateway daemon key ring file (the Java keystore created in “Create and populate the Gateway daemon key ring” on page 231). Figure 11-24 on page 235 shows the updated CICS Transaction Gateway panel of the configuration tool, which equates to the `PRODUCT` section of the configuration file.

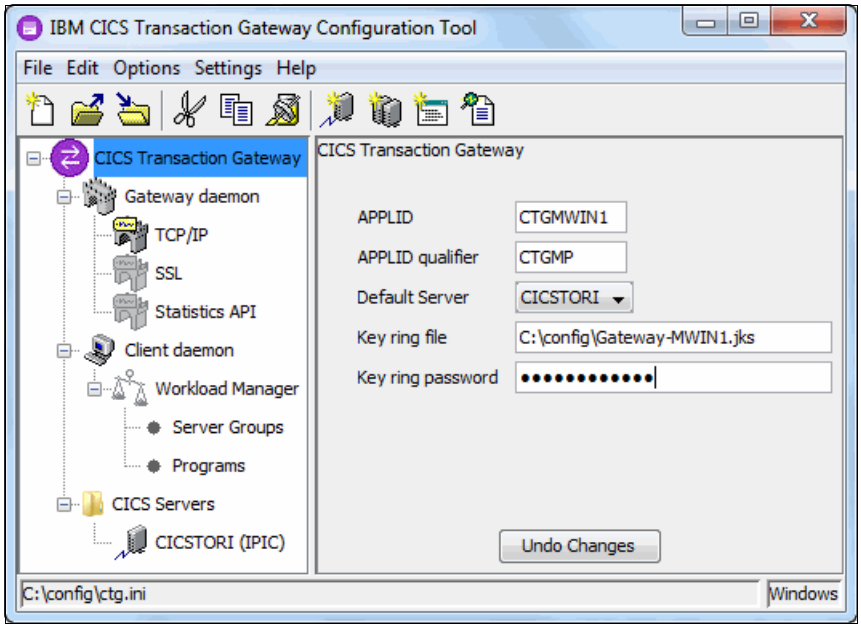


Figure 11-24 The CICS Transaction Gateway panel in the configuration tool with key ring details

The Key ring file and Key ring password fields must be completed if the SSL protocol is configured for client requests to the Gateway daemon, or if the SSL protocol is to be used for IPIC connections to CICS servers. Use the guidelines presented in Table 11-4.

Table 11-4 PRODUCT section parameters and values for the Gateway daemon key ring

Parameter	Value	Usage
Key ring file	`C:\Users\Administrator\Gateway-MWIN1.jks`	Java keystore file. Provide either the fully qualified path to the file or the file name if it is in the `<install_path>/bin` directory.
Key ring password	`cmVkYm9vawZ9`	Password for the Java keystore file. When the password is entered using the configuration tool, it is encrypted automatically.

Note: CICS TG V9.0 introduced SSL for IPIC connections in remote mode configurations. In previous releases, the key ring is defined by the SSL protocol handler in the `GATEWAY` section of the CICS TG configuration file. To use SSL for IPIC connections in V9.0, or later, the key ring file and key ring password are defined in the `PRODUCT` section of the CICS TG configuration file. The SSL protocol handler then shares the same key ring file attributes as any IPIC connections using SSL.

Figure 11-25 on page 236 shows the updated `PRODUCT` section of the CICS TG configuration file, including details of the Gateway daemon key ring.

```
SECTION PRODUCT
    APPLID=CTGMWIN1
    APPLIDQUALIFIER=CTGMP
    DEFAULTSERVER=CICSTORI
    KEYRING=C:\config\Gateway-MWIN1.jks
    KEYRINGPW=Y21Wa1ltOXZhd1o5
    KEYRINGPWSCRAMBLED=on
ENDSECTION
```

Figure 11-25 Updated PRODUCT section generated through the configuration tool

Note: The value of the `KEYRINGPW` attribute of the generated configuration file does not match the value of the password specified in the **`iKeyman`** tool and on the CICS Transaction Gateway panel of the CICS TG configuration tool. By default, the configuration tool generates an obfuscated representation of the key ring password and turns on the `KEYRINGPWSCRAMBLED` attribute.

The keystore, `Gateway-MWIN1.jks`, contains the personal certificate that identifies the Gateway daemon. This is a prerequisite for secure connectivity from remote applications, and secure IPIC connections with CICS servers.

Configure the SSL protocol handler

For remote applications to connect securely to the Gateway daemon, an SSL protocol handler is required. Figure 11-26 on page 237 shows the SSL settings panel that is used to create a protocol handler definition for the `GATEWAY` section of the configuration file.

Note: The key ring file and key ring password fields are pre-filled because they were already defined on the CICS Transaction Gateway settings panel for the `PRODUCT` section of the configuration file. The key ring configuration was moved to the `PRODUCT` section beginning with CICS TG V9.0 products, when SSL IPIC connections were introduced. Before V9.0, the Gateway daemon key ring was only used by the SSL protocol handler.

The SSL protocol handler shares a common set of attributes with a `tcp` protocol handler. Therefore, many of the attributes match the default values that are used in “Configure the tcp and statsapi protocol handlers” on page 215.

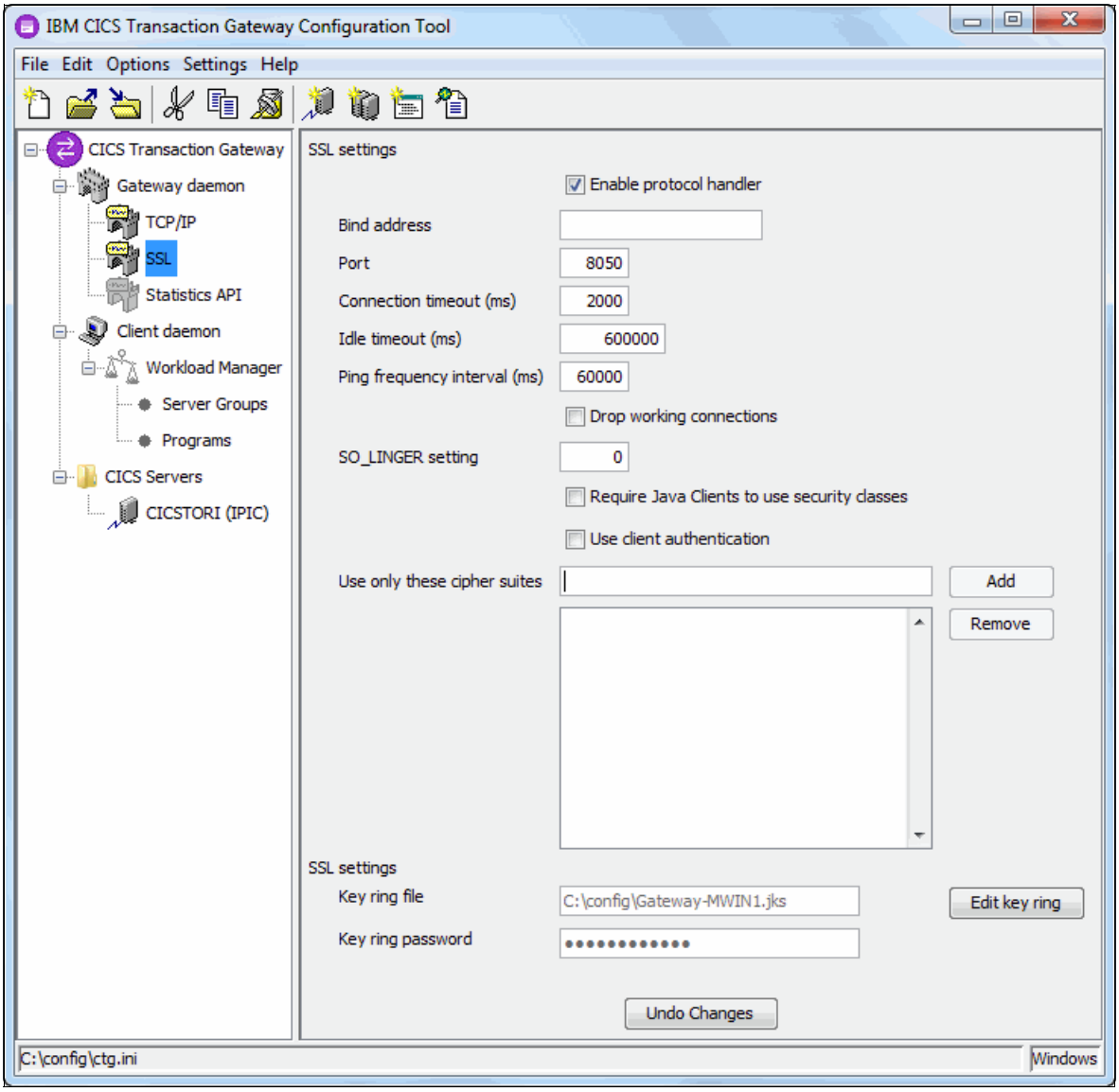


Figure 11-26 The SSL settings panel in the configuration tool for the SSL protocol handler

The parameters listed in Table 11-5 on page 238 apply to the SSL protocol handler. These parameters can be set in the configuration tool on the SSL settings panel, as shown in Figure 11-26, when the “Enable protocol handler” option is activated. Adding the SSL protocol handler also requires that the Gateway daemon key ring is configured (see “Configure the Gateway daemon to use the new keystore” on page 234).

Table 11-5 SSL protocol handler parameters

SSL settings field	Value	Usage
Use only these cipher suites	Optionally add a set of valid cipher suite names	Restrict the cipher suites available for use by remote applications connecting to the Gateway daemon SSL protocol handler
Key ring file	Value inferred from the `PRODUCT` section	Java keystore file
Key ring password	Value inferred from the PRODUCT section	Password for Java keystore file

If the “Use only these cipher suites” list contains entries, a remote application must use one of those cipher suites to connect to the Gateway daemon. If the remote application runtime environment does not support any of the cipher suites listed, it is unable to establish a connection. If the list contains no entries, a Java client application can connect using any available cipher suite.

When using the configuration tool, ensure that the “Enable protocol handler” setting is selected for each required protocol handler definition. This setting causes the protocol handler definitions to be added to the configuration file, as shown in Figure 11-27.

Note: The default ports for each protocol handler definition are `2006` for `tcp`, `8050` for `ssl`, and `2980` for `statsapi`.

```
protocol@tcp.handler=com.ibm.ctg.server.TCPHandler
protocol@tcp.parameters=connecttimeout=2000;idletimeout=600000;pingfrequency=60
000;port=2006;bind=;solinger=0;
protocol@ssl.handler=com.ibm.ctg.server.SslHandler
protocol@ssl.parameters=connecttimeout=2000;idletimeout=600000;pingfrequency=60
000;port=8050;bind=;solinger=0;
protocol@statsapi.handler=com.ibm.ctg.server.RestrictedTCPHandler
protocol@statsapi.parameters=port=2980;bind=;connecttimeout=2000;maxconn=5
```

Figure 11-27 The tcp, ssl, and statsapi protocol handler definitions, generated by the configuration tool

Verify the Gateway daemon initialization

Verify that the Gateway daemon initialized successfully. Verify that it is listening for *secure* application requests on the intended port number. Example 11-5 on page 239 shows selected log messages from the Gateway daemon initialization that indicate these outcomes.

Example 11-5 Gateway daemon initialization log messages for the SSL protocol handler

```
CTG8405I JSSE provider info: IBM JSSE2 7.0 Provider Implementation, IBM
Corporation, 7.0 build_20120529
CTG8489I The following cipher suites are provided by JSSE:
TLS_EMPTY_RENEGOTIATION_INFO_SCSV
    SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
    SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256
    SSL_RSA_WITH_AES_128_CBC_SHA256
    SSL_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
[Numerous additional ciphers deleted for space reasons]

CTG6524I Successfully started handler for the ssl: protocol on port 8050
```

With **ctgadmin**, you can verify the key ring file that is being used by a Gateway daemon by using the **ctginfo** option of the **dump** action:

```
ctgadmin -a dump -ctginfo
```

The CICS Transaction Gateway Informational Dump output is written to the Gateway daemon info log under log message `CTG8285I`. Check the value of the **KEYRING** parameter in the Gateway properties section.

If the key ring configuration is incorrect (for example, an incorrect password is specified), the Gateway daemon will fail to initialize and display this error log message:

```
CTG8491E Unable to read key ring C:\config\Gateway-MWIN1.jks reason =
java.io.IOException: Keystore was tampered with, or password was incorrect
```

11.5.2 Test the secure Gateway connection

The same Java sample applications that are used in 11.4, “Testing the Gateway daemon” on page 227 can also be used to verify the configuration of the Gateway daemon SSL protocol handler. However, a client application keystore file must first be created and configured to establish the secure connection.

Export the Gateway daemon’s public certificate

First, you need to extract the Gateway daemon’s public certificate so that it can be shared with clients that want to connect to the Gateway daemon. With the `gateway_mwin1_itso` Personal certificate highlighted in the Key database content panel, follow these steps:

1. Click **Extract Certificate**, which opens the dialog box shown in Figure 11-28 on page 240.
2. In the Data type drop-down list, select **Binary DER data**.
3. Enter the name and location of the output file to include the Gateway daemon public certificate. This example uses `gateway-mwin1-itso-cert.der`.
4. Select **OK**.

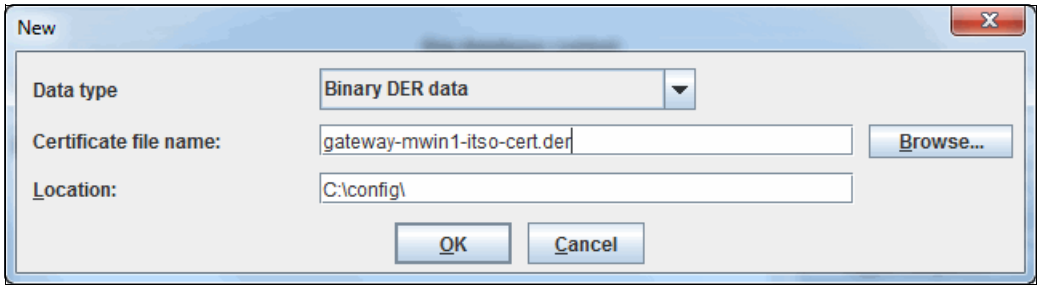


Figure 11-28 Extracting the public certificate for the Gateway daemon

Create and populate the client keystore

Note: Store the exported certificate in a safe place. It must be imported into a Java keystore that is used by remote applications that connect to this Gateway daemon using SSL.

For Java client applications to connect to the Gateway daemon using SSL, a Java keystore must be created for the client. The Gateway daemon's public certificate must then be added to the client's keystore as a signer (also known as a *trusted certificate*).

Follow these steps to import the Gateway daemon certificate to a client keystore:

1. Use the **`iKeyman`** tool to create a Java keystore (`.jks`) file on the client machine (for example, `C:\config\Client_MWIN1.jks`). Provide a password for the client application Java keystore. For the purpose of demonstration, we used the password `r3dbookClient`.
2. Select **Signer Certificates** from the Key database content drop-down list, as shown in Figure 11-29 on page 241, and then, click **Add**.

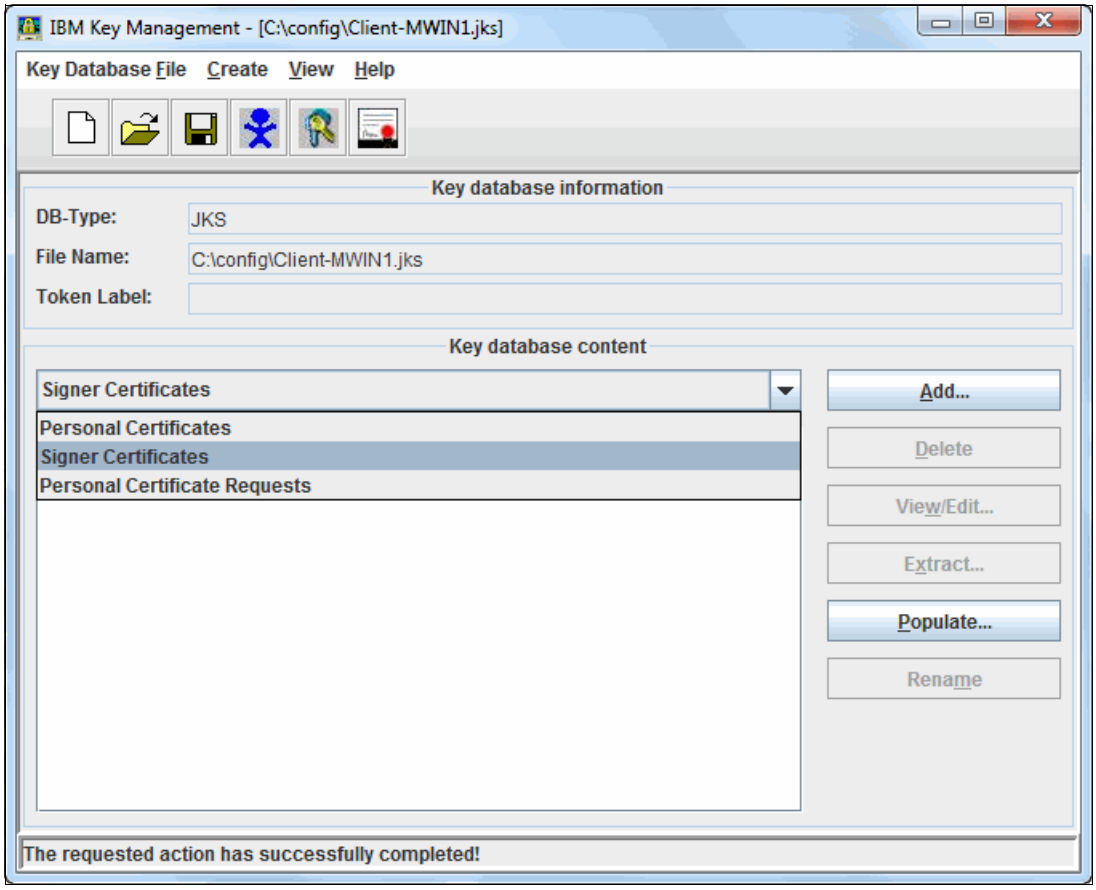


Figure 11-29 Selecting the type of certificate to add to the keystore

3. The Open dialog box, shown in Figure 11-30, is where you provide the location of the public certificate file to be imported. You can type the details or browse to select the Gateway daemon public certificate file, `gateway-mwin1-itso-cert.der`, which was exported in “Export the Gateway daemon’s public certificate” on page 239.

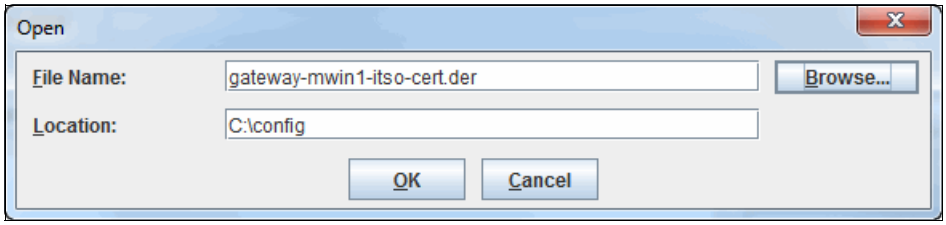


Figure 11-30 Adding the Gateway daemon signer certificate to the remote application Java keystore

4. Assign a label to the certificate to identify it in the keystore (see Figure 11-31 on page 242), and then, click **OK**.

Figure 11-31 Specifying a label for the signer certificate

Figure 11-32 shows the **iKeyman** tool's main view after the exported Gateway daemon certificate was added to the remote application Java keystore.

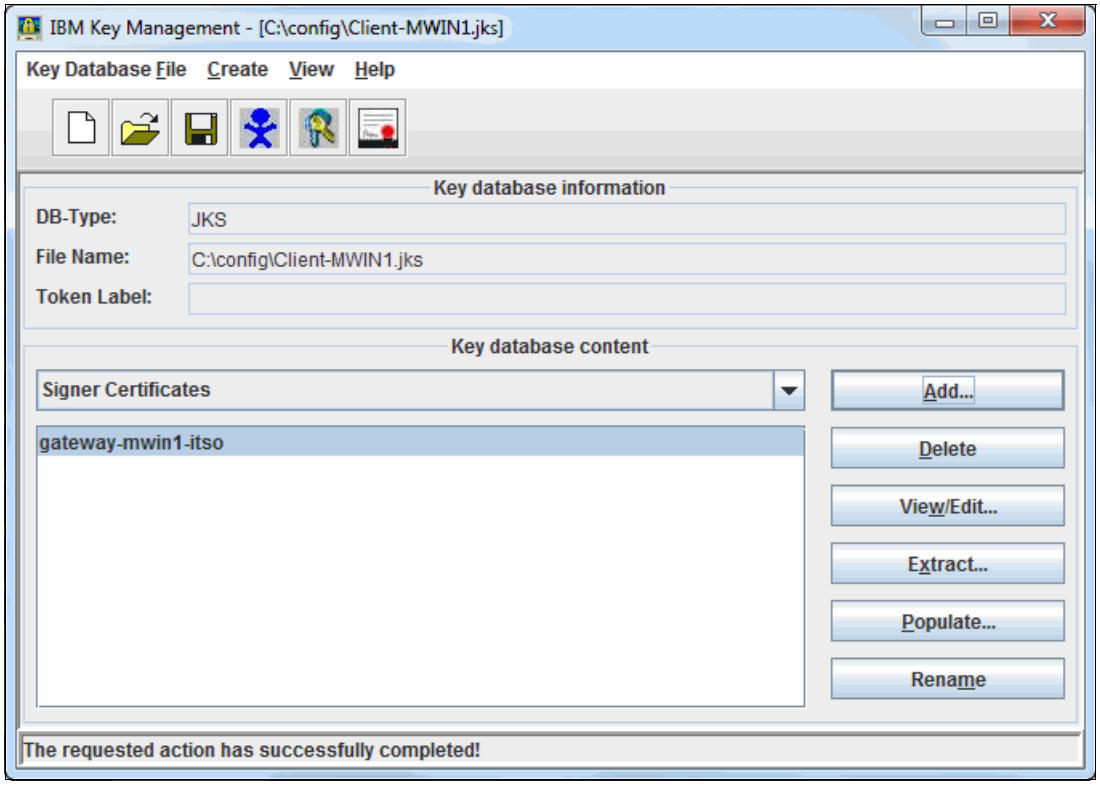


Figure 11-32 Gateway daemon signer certificate in the remote application keystore

Note: If an `ssl` protocol handler is configured to support *only* server authentication, you are now ready to use the secure connection between the remote application and the Gateway daemon. If the `ssl` protocol handler requires client authentication, you also need to follow the steps in 11.5.3, "Add client authentication" on page 244.

Verify the secure application connection

SSL connectivity between a client application and the Gateway daemon can be tested by using the sample Java application `EciB1`, as shown in Figure 11-33 on page 243. This example uses the keystore file (`Client-MWIN1.jks`) for the remote application and specifies its password (`r3dbookClient`).

```
C:\Users\Administrator>java com.ibm.ctg.samples.eci.EciB1 ssl://localhost 8050
C:\config\Client-MWIN1.jks r3dbookCl1ent
```

Figure 11-33 Testing the SSL connection with Java sample EciB1

If the test is successful, it returns the time and date from the host system as shown in the output in Figure 11-34.

```
CICS Transaction Gateway Basic ECI Sample 1

Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway URL]
                                          [Gateway port number]
                                          [SSL keyring]
                                          [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 localhost port:8050.

CICS servers defined:

         1. CICSTORI -CICS TOR region 1 IPIC


Choose server to connect to, or q to quit:
1

Program EC01 returned with data:-

        Hex: 32302f30362f31332030353a33323a32340
        ASCII text: 20/06/13 05:32:24
```

Figure 11-34 Simple EciB1 test output using the SSL protocol handler

You can verify secure connectivity between the remote application and the Gateway daemon by enabling connection logging, as described in “Gateway daemon log files” on page 212. Selecting the “Log Client connections and disconnections” option on the Logging tab of the Gateway daemon settings panel turns on the `connectionLogging` attribute in the `GATEWAY` section of the CICS TG configuration file.

Figure 11-35 shows the result of enabling connection logging when a secure connection is established between a remote application and the Gateway daemon.

```
[0] CTG6506I Client connected: [ConnectionManager-0] -
ssl@a98ae9c5[SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA:
Socket[addr=/127.0.0.1,port=57109,localport=8050]] using protocol TLSv1

[0] CTG6507I Client disconnected: [ConnectionManager-0] -
ssl@a98ae9c5[SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA:
Socket[addr=/127.0.0.1,port=57109,localport=8050]] using protocol TLSv1, reason = Java
client application closed the connection
```

Figure 11-35 Connection logging in the Gateway daemon for remote applications

11.5.3 Add client authentication

The scenario described in 11.5.2, “Test the secure Gateway connection” on page 239, enables the client application and the Gateway daemon to communicate using SSL server authentication. However, the authentication handshake is one-sided because only the identity of the server is authenticated.

To allow a two-way handshake, client authentication needs to be enabled so that both the client application and Gateway daemon personal certificates are exchanged. An additional self-signed personal certificate must be created in the remote application Java keystore. This self-signed client application certificate, in turn, is extracted as a public certificate so that it can be added to the Gateway daemon’s Java keystore as a signer (or trusted) certificate.

This process closely matches the pattern that we applied to configure server authentication using **`iKeyman`**, which is described in 11.5.2, “Test the secure Gateway connection” on page 239. Follow these steps:

1. In the **`iKeyman`** tool, open the remote application Java keystore that you created in “Create and populate the client keystore” on page 240. In this configuration, it is the file `C:\config\Client-MWIN1.jks`.
2. Select **Personal Certificates** from the Key database content drop-down list, if it is not already selected.
3. Click **New Self-Signed** to open the Create New Self-Signed Certificate dialog box (Figure 11-36).

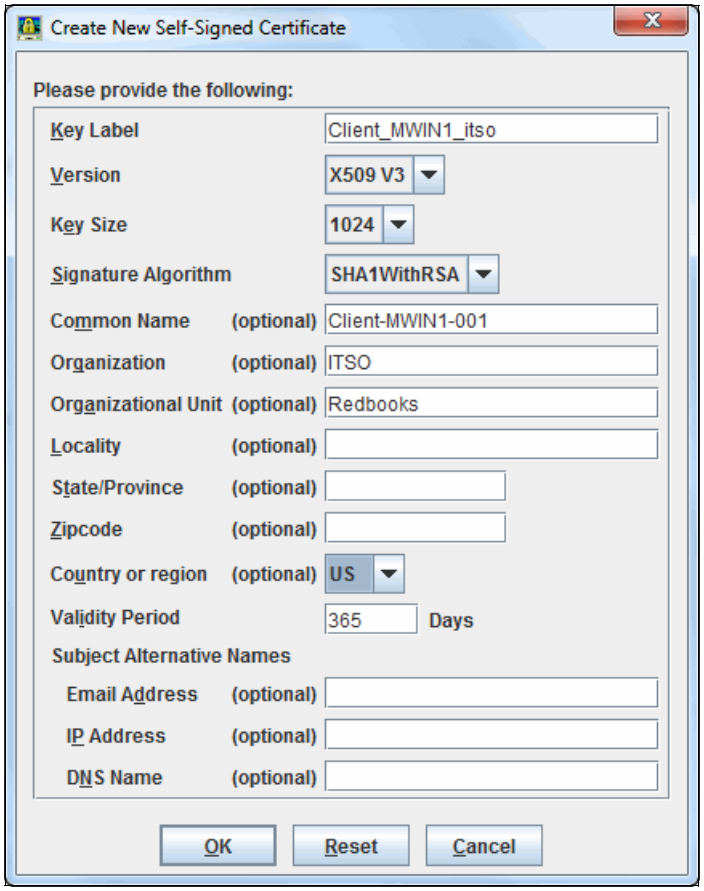


Figure 11-36 Creating a new self-signed personal certificate for the remote application Java keystore

4. Enter the appropriate details for the client application self-signed personal certificate, click **OK**, and then, save the Java keystore. In this example, the value specified for the Key label field is `Client_MWIN1_itso`, and the common name is an identifier unique to the organization that refers to this specific client application instance, rather than to a specific machine.

   The value of the Key label field is used to provide a meaningful name to the certificate. It can be used as a primary identifier for the certificate in other Java keystore tools, such as **`keytool`**.

5. In the IBM Key Management (**`iKeyman`**) main view, click **Extract... Certificate** to extract the public certificate, selecting **Binary DER** in the Data type field. In this example, the target file is `C:\config\Client-mwin1-itso-cert.der`.

6. Switch back to the Gateway daemon Java keystore. In this example, open the Java keystore file `C:\config\Gateway-MWIN1.jks`.

7. Select **Signer Certificates** from the Key database content drop-down list, and then, click **Add**. Select the remote application public certificate file (`C:\config\Client-mwin1-itso-cert.der`) and provide a suitable label. In this example, the label is `client-mwin1-001`.

8. Save the Gateway daemon Java keystore.

The Gateway daemon Java keystore and the client application Java keystore now contain the resources required to use SSL client authentication.

An additional change is required to enable client authentication for the `ssl` protocol handler. Go to the CICS TG configuration tool, shown in Figure 11-37 on page 246, and select **Use client authentication**.

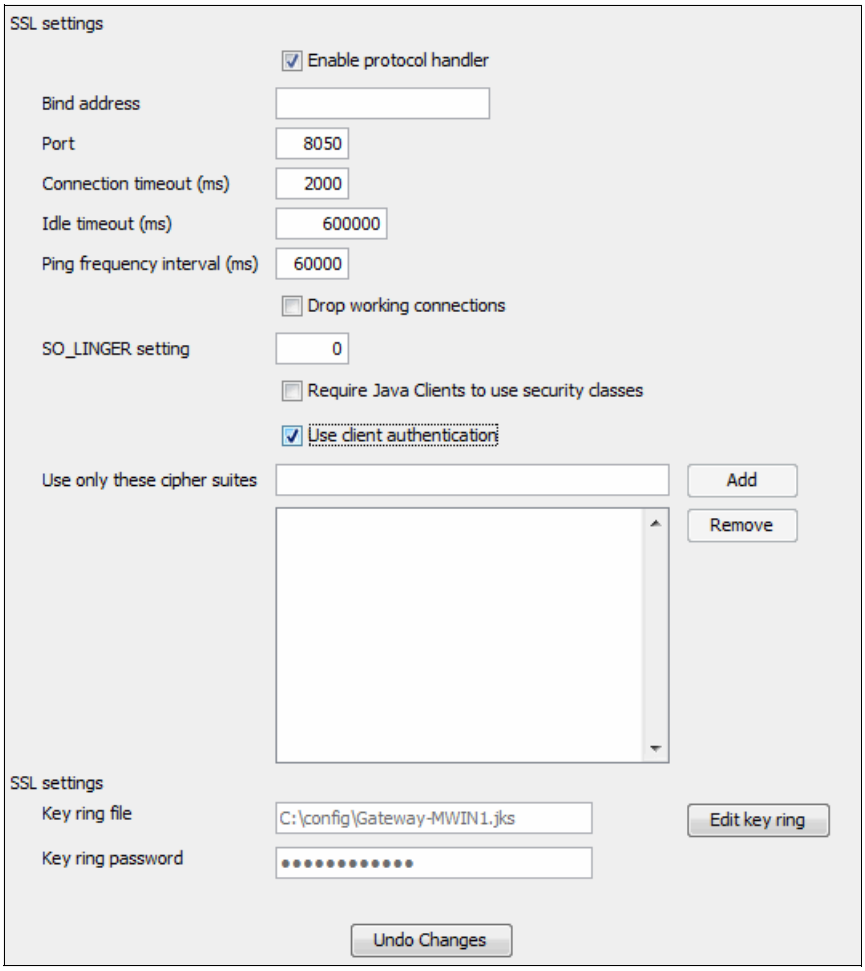


Figure 11-37 Enabling client authentication on the SSL settings panel in the configuration tool

Selecting this option turns on the `clientauth` attribute in the SSL protocol handler definition of the CICS TG configuration file, as shown in Figure 11-38.

```
protocol@ssl.handler=com.ibm.ctg.server.SslHandler
protocol@ssl.parameters=clientauth=on;connecttimeout=2000;idletimeout=600000;pingfrequen
cy=60000;port=8050;bind=;solinger=0;
```

Figure 11-38 SSL protocol handler definition with client authentication enabled

For more details, search for the CICS TG Information Center topic, *Configuring SSL*:

http://ibm.co/1p38yCv

The CICS TG Information Center also contains useful scenario documentation, including the topic, *Configuring SSL security between a Java Client and the Gateway daemon (SC06)*.

Verify the client authentication configuration

The Java sample application `EciB1` can be used to verify SSL connectivity with client authentication between the client application and the Gateway daemon. Figure 11-39 shows the command used to run `EciB1` with the required parameters. A secure connection is attempted when the **`Gateway URL`** parameter includes the prefix **`ssl://`** with the port number for the `ssl` protocol handler (8050), the location of the client application Java keystore file, and the password for that Java keystore.

```
C:\Users\Administrator>java com.ibm.ctg.samples.eci.EciB1 ssl://localhost 8050
C:\config\Client-CTGMWIN1.jks r3dbookCl1ent
```

Figure 11-39 Testing the SSL connection with Java sample application EciB1

If this application runs successfully, it returns the time and date from the host system, as shown in the output in Figure 11-40.

```
CICS Transaction Gateway Basic ECI Sample 1

Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway URL]
                                          [Gateway port number]
                                          [SSL keyring]
                                          [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://localhost port:8050.

CICS servers defined:

         1. CICSTORI -CICS TOR region 1 IPIC


Choose server to connect to, or q to quit:
1

Program ECO1 returned with data:-

        Hex: 32302f30362f31332030363a31303a30390
        ASCII text: 20/06/13 06:10:09
```

Figure 11-40 Simple EciB1 test output using the SSL protocol handler and client authentication

Note: This test can be important because CICS TG offers no log messages or system statistics to verify that a Gateway daemon is using client or server authentication with the `ssl` protocol handler. However, remote applications with a public certificate that is not trusted by the Gateway daemon will fail to connect.

11.5.4 Secure the IPIC connection

To allow secure SSL connectivity with a CICS TS region (or TXSeries region), the Gateway daemon must be able to authenticate the personal certificate of the partner system. Typically, the public certificate authority (CA) certificates that are preloaded into a keystore are sufficient when the personal certificate is signed by a public certificate authority, such as VeriSign. In this part of the book, self-signed certificates are created by using a generated certificate authority, rather than a public certificate authority.

In 9.4.1, “Create a key ring for CICS” on page 163, we create our own certificate authority, `CICS-Sample-Certification`, and use it to sign the personal certificate of the CICS region, `CICSTOR1`. To configure the Gateway daemon to use SSL server authentication on an IPIC connection with `CICSTOR1`, the CA signer certificate must be imported to the Gateway daemon Java keystore file.

The steps to obtain the CA signer certificate are described in “Export the CA signer certificate for keytool” on page 190. The CA signer certificate file that is exported from RACF, `cics.ca.cer`, can be transferred from z/OS to the Windows server by using binary mode FTP.

Adding the CA signer certificate to the Gateway daemon keystore

Follow these steps to use the **`iKeyman`** utility to import the CA signer certificate to the Gateway daemon keystore:

1. Launch **`iKeyman`** and open the Java keystore for the Gateway daemon. In this example, the Java keystore file is `Gateway-MWIN1.jks`.
2. Select **Signer Certificates** from the Key database content drop-down list, as shown in Figure 11-41, and then, click **Add** and browse to the location of the CA certificate that you transferred using FTP.

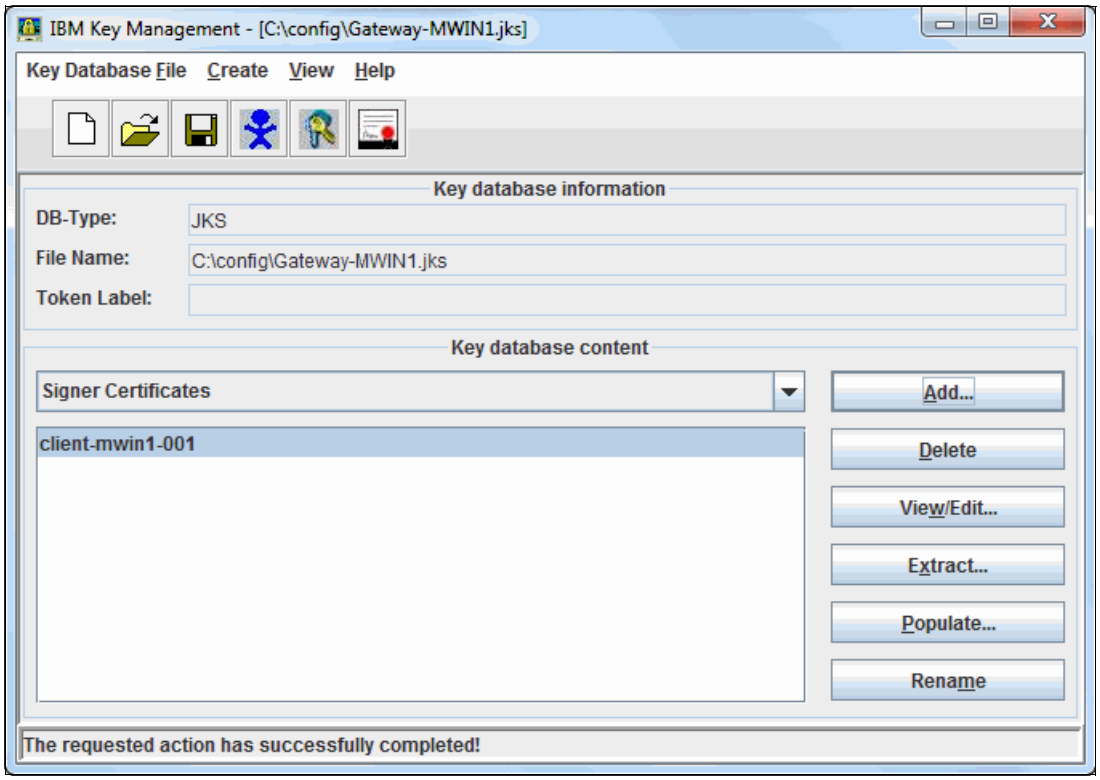


Figure 11-41 Adding a signer certificate to the Gateway daemon Java keystore

3. The Open dialog box, shown in Figure 11-42, asks for details about the location of the CA signer certificate file that you are importing. Type the details or browse to find and select the CA signer certificate file. In this example, the file is `cics.ca.cer`.

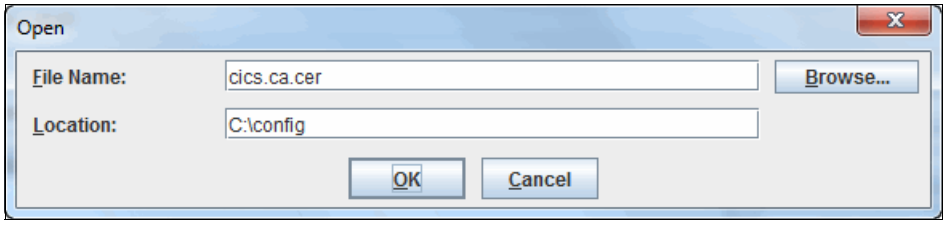


Figure 11-42 Details for CA signer certificate being added to Gateway daemon Java keystore

4. Give the certificate an identifying label, as shown in Figure 11-43, and then, click **OK**.

Figure 11-43 Labeling the CA signer certificate

The CA certificate is now included in the set of signer certificates in the Gateway daemon keystore, as shown in Figure 11-44.

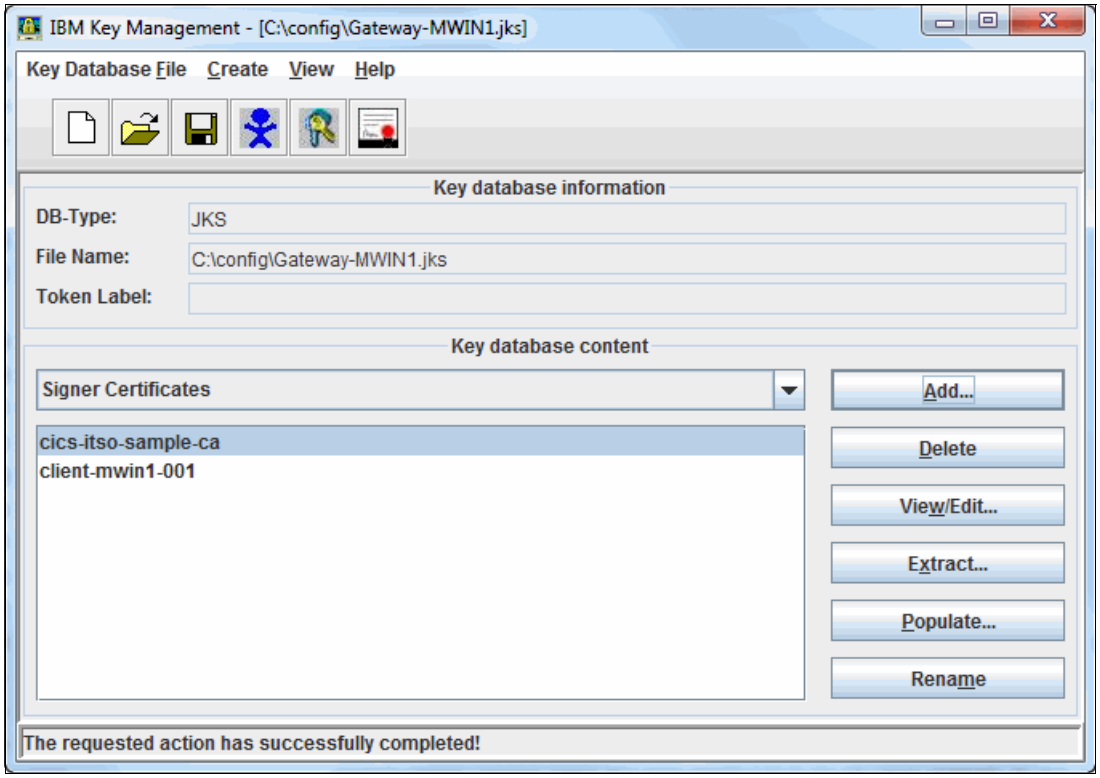


Figure 11-44 Signer certificates in the Gateway daemon Java keystore

After saving the Gateway daemon Java keystore, it is ready to be used for an IPIC SSL connection to the CICS `TCPIPService` `GW1IPICS`, which was defined in 9.4.3, “Configure a TCPIPSERVICE for SSL” on page 165. The `SSl` attribute of the CICS TS `TCPIPService` definition determines whether server or client authentication is required on this IPIC connection.

Note: SSL connectivity to a CICS server is supported only when using the IPIC protocol.

Configure the secure IPIC server connection

The Gateway daemon is now configured to authenticate the identity of the partner CICS region, which has a personal certificate that is signed by the CA signer certificate, `CICS-Sample-Certification`. Here, we create a secure IPICSERVER definition using server authentication using the CICS TG configuration tool (**`ctgcfg`**).

In the navigation tree, right-click the **CICS Servers** node, and then, select **New Server**. The attributes for this CICS server definition are similar to those for the non-SSL IPIC server connection that we defined in 11.2.5, “The IPICSERVER section” on page 216, except that this one uses port number `9001` for the CICS `TCPIPService` `GW1IPICS`, and you must select the **Use SSL** option.

Figure 11-45 on page 251 shows the CICS server connection, `CICSTORS`, defined to use IPIC SSL.

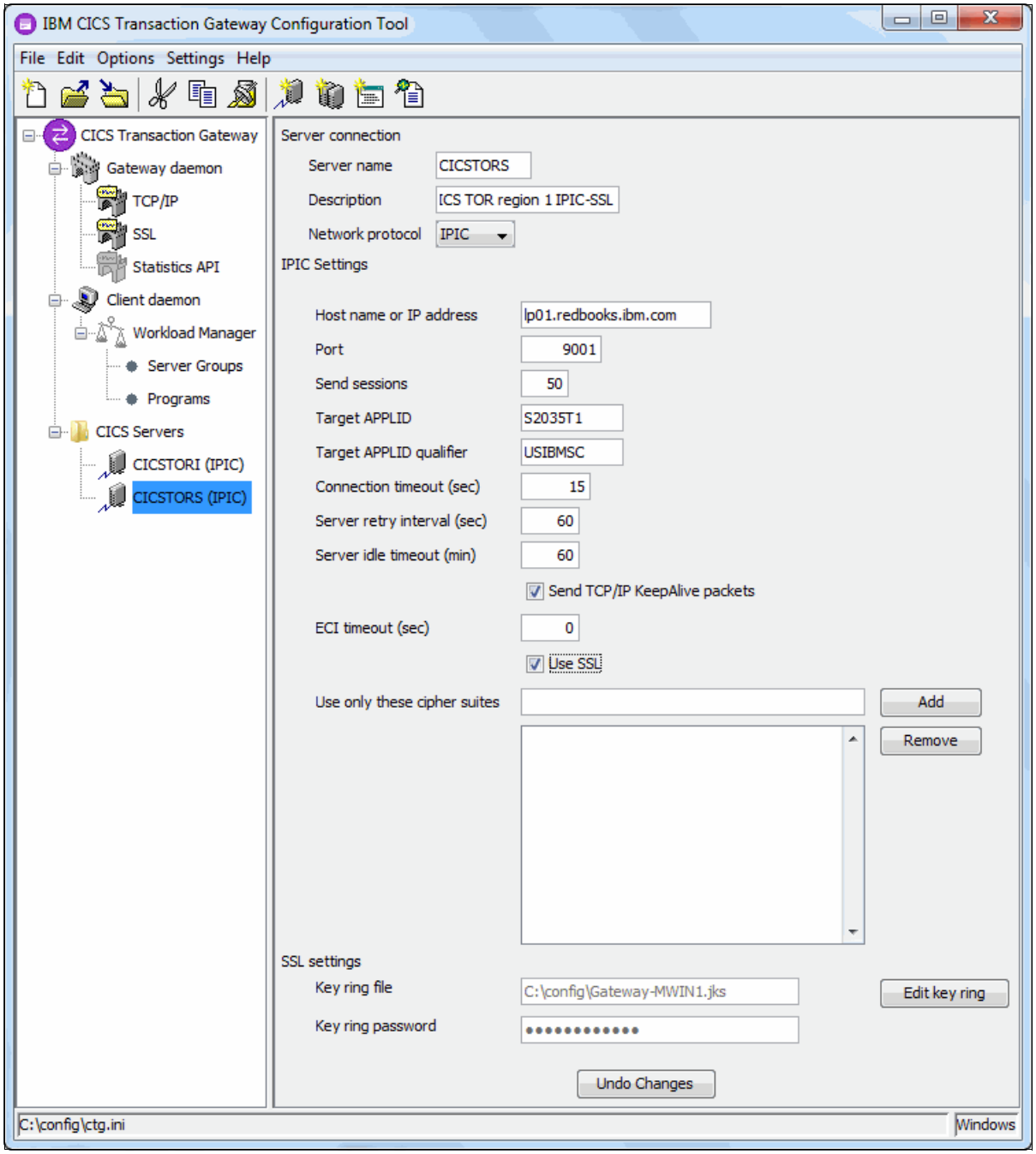


Figure 11-45 Server connection panel showing new CICS server definition for IPIC SSL connection CICSTORS

Each `IPICSERVER` definition in the CICS TG configuration file includes a parameter to control the use of SSL. Figure 11-46 on page 252 shows the `IPICSERVER` definition in the configuration file as it was generated by the configuration tool, including the `SSL` attribute with the value `Y`.

```
SECTION IPICSERVER = CICSTORS
    DESCRIPTION=CICS TOR region 1 IPIC-SSL
    SRVIDLETIMEOUT=0
    HOSTNAME=lp01.redbooks.ibm.com
    PORT=9001
    CONNECTTIMEOUT=0
    SRVRETRYINTERVAL=60
    TCPKEEPALIVE=Y
    SENDSESSIONS=50
    ECITIMEOUT=0
    CICSAPPLID=S20351T1
    CICSAPPLIDQUALIFIER=USIBMSC
    SSL=Y
ENDSECTION
```

Figure 11-46 The IPICSERVER section for CICSTORS generated by the configuration tool

Table 11-6 lists the attributes of an IPIC SSL connection definition, including the values used for the connection to the `TCPIPService GW1IPICS`, in the CICS region `CICSTOR1`.

Table 11-6 IPICSERVER definition for CICSTORS

Configuration parameter	Value	Usage
Server name	`CICSTORS`	Use this name for all requests to access the server from remote applications.
Description	`CICS TOR region 1 IPIC-SSL`	Description for the server (optional, but useful to identify this server definition).
Hostname	`lp01.redbooks.ibm.com`	Host name for the server.
Port	`9001`	Port number on which the target CICS server is listening.
Send sessions	`50`	Number of simultaneous transactions or CICS tasks that are allowed over the connection (limited by the ReceiveCount defined on the IPCONN definition).
Target APPLID	`S20351T1`	APPLID of the target CICS server (optional unless the CICSAPPLIDQUALIFIER is specified, in which case the CICSAPPLID must match the APPLID of the target CICS server).
Target APPLID qualifier	`USIBMSC`	Optional unless CICSAPPLID is specified (in which case the CICSAPPLIDQUALIFIER must match the network ID of the target CICS server).
Use SSL	`Y`	A value of `Y` Indicates that the SSL protocol is to be used on this IPIC connection. Enabling SSL requires a key ring file to be defined in the `PRODUCT` section.

Configuration parameter	Value	Usage
Use only these cipher suites	Required ciphers or blank	Specifies the cipher suites that CICS TG can use to connect to this CICS region. Cipher suites must be supported by the Java Secure Socket Extension (JSSE) provider used by CICS TG. If this parameter is omitted, all available cipher suites can be used.

The Gateway daemon does not produce log messages indicating the use of SSL for IPIC connections during initialization, or when the connection is established. However, the set of statistics associated with each IPIC connection can be queried to verify the use of SSL.

Verify the client authentication configuration

This subsection uses the Java sample application `EciB1` to make an ECI request to `CICSTOR1`, using the secure IPIC connection `CICSTORS`. It also uses a secure connection between the remote application and the Gateway daemon. It shows the log messages and statistics associated with the secure IPIC connection.

Figure 11-47 shows `EciB1` with parameters to specify the SSL protocol for the Gateway URL, the Gateway daemon SSL protocol handler port (`8050`), the Java keystore file for the remote application, and the password for that Java keystore.

```
C:\Users\Administrator>java com.ibm.ctg.samples.eci.EciB1 ssl://localhost 8050
C:\Users\Administrator\CTGMWIN1_client_key.jks redbooks
```

Figure 11-47 Testing the SSL connection with sample application EciB1

In this case, the secure connection to `CICSTOR1` is chosen. If this application runs successfully, it returns the time and date from the host system, as shown in the output in Figure 11-48 on page 254.

```
CICS Transaction Gateway Basic ECI Sample 1

Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway URL]
                                          [Gateway port number]
                                          [SSL keyring]
                                          [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://localhost port:8050.

CICS servers defined:

         1. CICSTORI -CICS TOR region 1 IPIC
         2. CICSTORS -CICS TOR region 1 IPIC-SSL


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

Program ECO1 returned with data:-

        Hex: 32302f30362f31332030363a31303a30390
        ASCII text: 20/06/13 06:10:09
```

Figure 11-48 Output verifying end-to-end secure connectivity

Gateway daemon log messages (Example 11-6), CICS log messages, and CICS resource inquiry commands (**CEMT INQuire**) *do not* reveal that an IPIC connection is using SSL. However, the CICS TG statistic `CSx_PROTOCOL` (where *x* is the name of the IPICSERVER definition) indicates a value of either `IPIC` or `IPICSSL`. This statistic can be inspected by using the command-line utility **ctgadmin**, or from the CICS connections view of the CICS TG perspective of the CICS Explorer.

Example 11-6 Gateway daemon log messages showing establishment of an IPIC connection

```
02/07/14 13:20:47:661 [0] CTG8477I About to connect to server CICSTORS
02/07/14 13:20:48:391 [0] CTG8429I Established new IPIC connection to CICS server
CICSTORS with: negotiated session limit=50, CICSAPPLID=CICSPAZ8
CICSAPPLIDQUALIFIER=USIBMSC , HOSTNAME=lp01.redbooks.ibm.com, PORT=9001, sockets=2
```

Example 11-7 shows the output of the following request to display the statistics for the CICS server `CICSTORS`, which we are filtering for so-called Startup statistics (note the "**s**").

```
ctgadmin -a stats -gs=CSCICSTORS -st=s
```

Example 11-7 Output of a request to display startup statistics for CICS server CICSTOR1

```
CTG8239I Response received from CICS Transaction Gateway
CSCICSTORS - CICS Server (instance)
   CSCICSTORS_SPROTOCOL=IPICSSL (CICS server protocol)
   CSCICSTORS_SIPADDR=lp01.redbooks.ibm.com (CICS server TCP/IP address)
   CSCICSTORS_SIPPORT=9001 (CICS server TCP/IP port)
   CSCICSTORS_SSESSMAX=100 (Number of requested IPIC sessions)
```

The combination of the CSx_SPROTOCOL statistic and the Gateway daemon log message CTG8429I verifies that the IPIC connection CICSTORS is successfully established and that it is secure.

11.6 Client daemon configuration

The Client daemon component is not required for IPIC-only configurations. However, if you require connectivity based on either SNA APPC or the CICS Family TCP/IP protocol, the Client daemon must be configured.

This section describes the additional configuration steps that are required to use the CICS Family TCP/IP protocol, but it does not go into significant detail. IPIC is the strategic option for CICS Family intercommunications.

Solutions based on SNA APPC and the CICS Family TCP/IP protocol are described in an earlier publication, *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133, and in the CICS TG for Multiplatforms Information Center:

http://pic.dhe.ibm.com/infocenter/cicstgmp/v9r0/index.jsp

To configure the Client daemon with a TCP/IP or SNA APPC connection, these items must be present in the CICS TG configuration file:

- The CLIENT section

  Defines general properties of the Client daemon
- A DRIVER section

  Defines the TCPIP or SNA protocol driver
- A SERVER section

  Defines at least one CICS server

As with the Gateway daemon, the Client daemon can be configured by using the CICS TG configuration tool. The Client daemon configuration is stored in the CICS TG configuration file, together with the PRODUCT and GATEWAY sections that are described in 11.2, “Configure the Gateway daemon” on page 207. Like the Gateway daemon, the Client daemon uses the value of the **CICSCLI** environment variable to determine the location of the CICS TG configuration file.

11.6.1 The CLIENT section

The CLIENT section of the configuration file defines the operating parameters and log file locations for the Client daemon. You use the Client daemon configuration panel in the configuration tool to specify the attributes for the Client daemon. Because this book is more focused on IPIC, each attribute is not examined in detail here.

Figure 11-49 on page 256 shows the Resources tab of the Client daemon configuration panel. We reviewed and accepted each of the default values suggested by the CICS TG configuration tool.

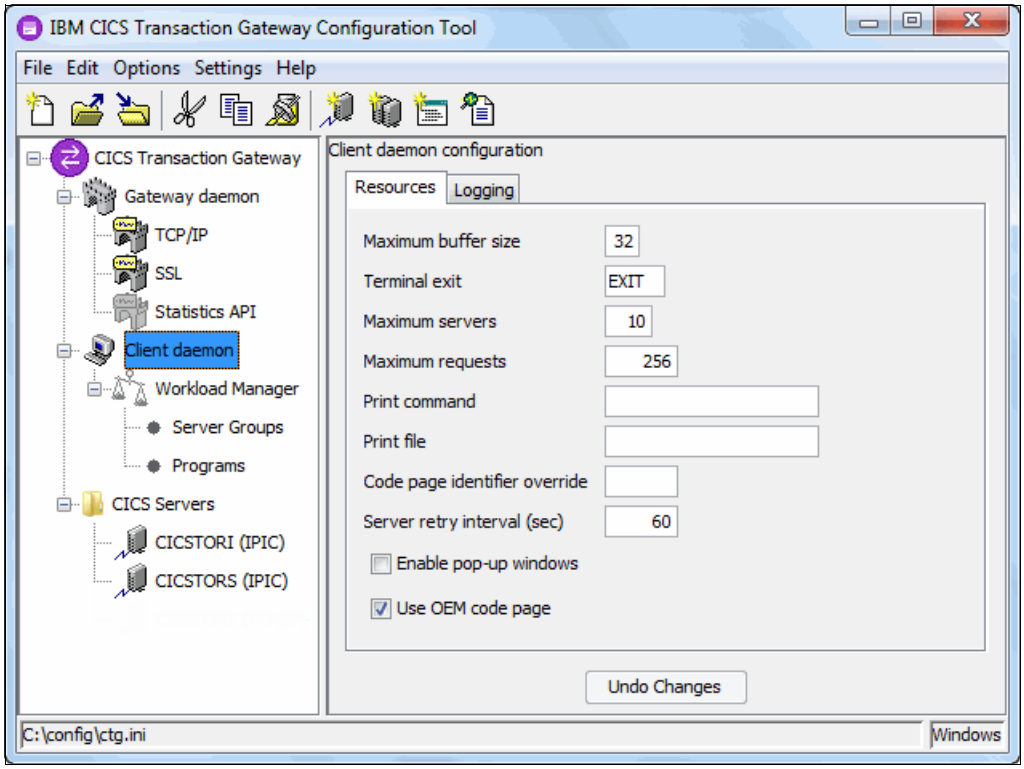


Figure 11-49 Resources tab of the Client daemon configuration panel in CICS TG configuration tool

Note: For the CICS TG Desktop Edition product, the Client daemon is limited to processing a maximum of 32 concurrent requests.

Figure 11-50 shows the Logging tab of the Client daemon configuration panel. We reviewed and accepted each of the default values suggested by the CICS TG configuration tool.

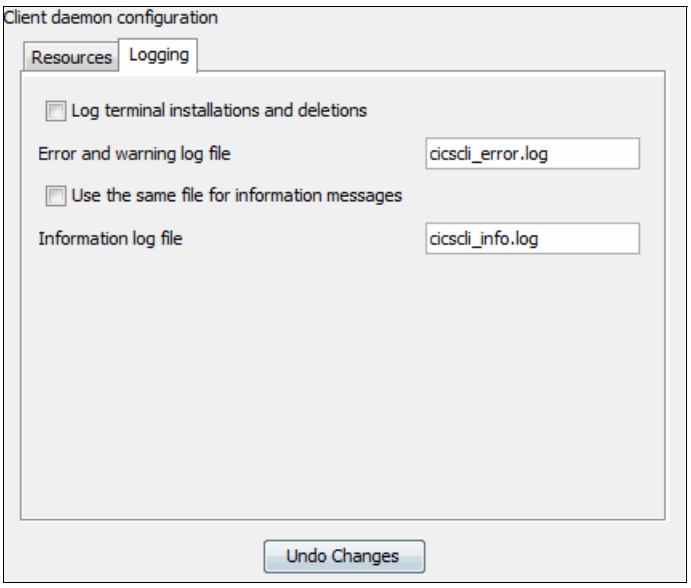


Figure 11-50 Logging tab of the Client daemon configuration panel in the CICS TG configuration tool

By default, all log files are written to the product data path on Windows platforms, defined by the environment variable, **ProgramData**, or the /var/cicscli directory on UNIX and Linux platforms.

Figure 11-51 shows the CLIENT section, as generated by the configuration tool.

```
SECTION CLIENT = *
    ENABLEPOPUPS=N
    TERMINSTLOGGING=N
    LOGFILE=cicscli_error.log
    LOGFILEINFO=cicscli_info.log
    MAXBUFFERSIZE=32
    MAXREQUESTS=256
    MAXSERVERS=10
    MAXWRAPSIZE=0
    SRVRETRYINTERVAL=60
    TERMINALEXIT=EXIT
    TRACEFILE=cicscli.bin
    USEOEMCP=Y
    LOADMANAGER=N
ENDSECTION
```

Figure 11-51 The CLIENT section, as generated by the configuration tool

Note: You can set the **MAXWRAPSIZE** and **TRACEFILE** parameters by selecting **Configuration tool** → **Options** → **Trace settings**.

11.6.2 The DRIVER section

The DRIVER section of the CICS TG configuration file defines the CICS connection protocols available for use by the Client daemon. The required DRIVER sections are automatically generated when using the CICS TG configuration file. Here, we explain the attributes and possible values of the DRIVER section.

Example 11-8 shows the syntax of a DRIVER section in the configuration file.

Example 11-8 The syntax of the DRIVER section in the CICS TG configuration file

```
SECTION DRIVER = <TCPIP | SNA>
    DRIVERNAME=<driver DLL identifier>
ENDSECTION
```

The appropriate values are shown for each protocol for the DRIVERNAME attribute:

- SNA driver:
  - CCLWNTSN for Windows operating systems
  - CCLIBMSN for UNIX and Linux operating systems
- CICS Family TCP/IP protocol:
  - CCLWNTIP for Windows operating systems
  - CCLIBMIP for UNIX and Linux operating systems

Figure 11-52 on page 258 shows the DRIVER section of the CICSTG configuration file for the CICS Family TCP/IP protocol, as generated by the configuration tool for the Windows platform.

```
SECTION DRIVER = TCPIP
    DRIVERNAME=CCLWNTIP
ENDSECTION
```

Figure 11-52 The DRIVER section for the CICS Family TCP/IP protocol

11.6.3 The SERVER section

Each `SERVER` section defines the properties of a connection to a CICS server, using one of the network protocols available with the Client daemon (`SNA` or `TCP/IP`). Here, we create a new CICS server definition, `CICSTORT`, using the CICS Family TCP/IP protocol.

Use the CICS TG configuration tools (Figure 11-53) to add a new CICS server definition using the CICS Family TCP/IP protocol.

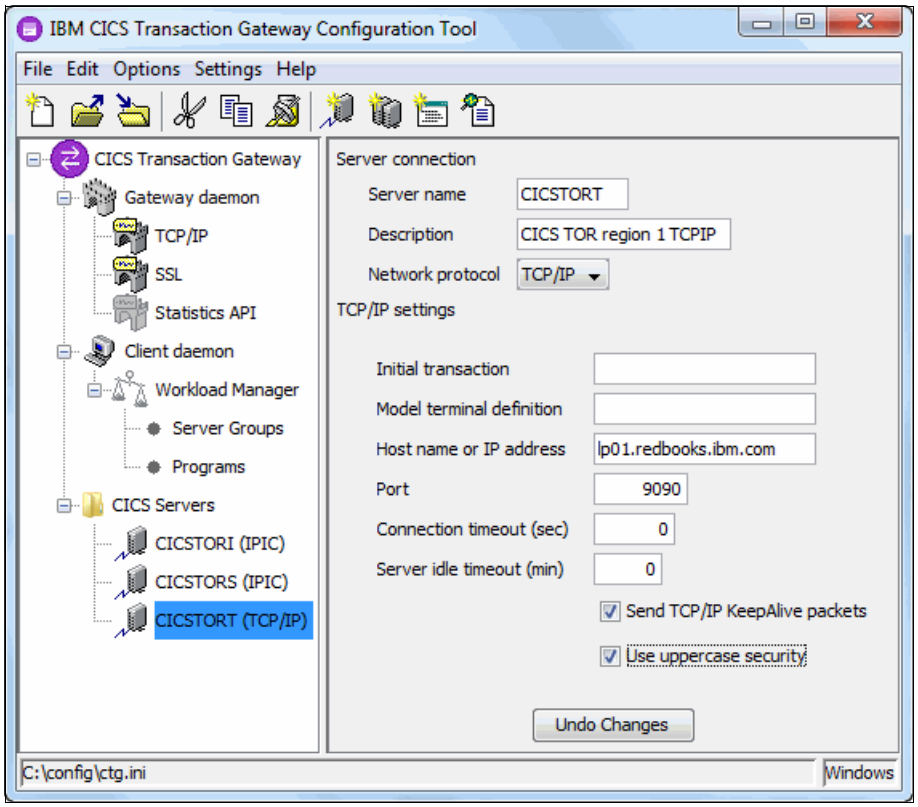


Figure 11-53 Adding a new CICS server definition using the CICS Family TCP/IP protocol

Figure 11-54 on page 259 shows the `SERVER` section for the CICS Family TCP/IP connection to `CICSTOR1`, as generated by the configuration tool.

```
SECTION SERVER = CICSTORT
    DESCRIPTION=CICS TOR region 1 TCP
    SRVIDLETIMEOUT=0
    REGION=SRVRWLM
    UPPERCASESECURITY=Y
    PROTOCOL=TCPIP
    NETNAME=lp01.redbooks.ibm.com
    PORT=9090
    CONNECTTIMEOUT=0
    TCPKEEPALIVE=Y
ENDSECTION
```

Figure 11-54 The SERVER section for CICSTORT, as generated by the configuration tool

Note: The **REGION** parameter is added if the CICS server is defined in a Workload Manager Server Group. For more information, see 11.6.4, “The LOADMANAGER section (Windows)” on page 259.

For additional details about the SERVER definition shown in Figure 11-54, see the topic, *Configuring CICS server connections*, in the CICS Transaction Gateway for Multiplatforms Information Center:

http://ibm.co/1nGgBB0

Table 11-7 lists the parameters for the SERVER section.

Table 11-7 SERVER parameters for CICSTORT

Configuration parameter	Value	Usage
Server name	CICSTORT	Use this name for all requests to access the server from client applications.
Description	CICS TOR region 1 TCP	Optional, but useful to identify this server definition.
Network protocol	TCP/IP	The communication protocol being used.
Port	9090	The server port number to which the Client daemon connects.
Host name or IP address	lp01.redbooks.ibm.com	The character-based or numeric TCP/IP identifier for the host on which the CICS server is running.

11.6.4 The LOADMANAGER section (Windows)

The CICS TG for Multiplatforms and CICS TG Desktop Edition products include a feature for load balancing, which is called *Workload Manager* (WLM), that is exclusive to the Windows operating systems. The LOADMANAGER section of the configuration file is generated by the configuration tool based on attributes that are set on the Workload Manager settings panel.

The WLM feature is designed to provide both round-robin and weight-biased workload distribution for Client daemon-based CICS server connections. The feature can be set to influence the distribution of work by both program ID and user-defined server groups.

Because the Workload Manager feature is part of the Client daemon component, IPIC-connected servers cannot be included in the `LOADMANAGER` section of the configuration file. However, the Gateway daemon CICS Request Exit provides an alternative facility for workload distribution. It is available for use with any of the CICS communications protocols supported by CICS TG products, and on all platforms.

For additional details about using the WLM feature on Windows systems, see the *Windows Workload Manager* section of the CICS Transaction Gateway for Multiplatforms Information Center:

http://ibm.co/1nkLQqV

12

Integrating with WebSphere Application Server

This chapter describes the use of WebSphere Application Server as an example of integrating a Java Platform, Enterprise Edition (Java EE) application server with CICS Transaction Server (CICS TS) through CICS Transaction Gateway (CICS TG). It includes key considerations for using the CICS resource adapters in the following topologies:

- Local and remote mode
- Non-transactional and extended architecture (XA) transactions
- Non-secure and Secure Sockets Layer (SSL) scenarios

Additionally, it describes how to install the CICS TG ECI resource adapter and configure a JCA connection factory to use Internet Protocol interconnectivity (IPIC) connections to CICS TS in both local and remote mode. Finally, it uses the JCA installation verification test (`IVT`) to test the end-to-end flow.

This chapter includes the following topics:

12.1 Java EE application servers supported by CICS TG

The CICS TG JCA resource adapters are compatible with a number of Java EE application servers:

- WebSphere Application Server for Multiplatforms
- WebSphere Application Server for z/OS
- Java EE certified application servers that run the JCA resource adapter installation verification test (`IVT`) successfully, including all non-IBM application servers, such as Oracle WebLogic, Glassfish, JBoss, and others

> **Important:** The `IVT` that is provided with CICS TG must run successfully before problems can be reported to IBM.

For information about the supported Java EE application server versions, available protocols, and transaction support, see the CICS TG for Multiplatforms Information Center:

http://ibm.co/1nVOQuI

> **Support note:** This information does not apply to CICS Transaction Gateway Desktop Edition.

12.2 The CICS resource adapters

CICS TG provides the following JCA resource adapters:

- The ECI resource adapter (`cicseci.rar`) provides a high-level common client interface (CCI) to the ECI for sending ECI requests to CICS servers.
- The external presentation interface (EPI) resource adapter (`cicsepi.rar`) provides a high-level CCI interface to the EPI to install terminals and run 3270-based transactions on a CICS server.

> **Note:** Before CICS TG Version 8.1, two versions of the ECI resource adapter were provided: `cicseci.rar` (for local transaction support) and `cicseciXA.rar` (for global transaction support).

JCA resource adapters implement qualities of service through the following system contracts: connection management, transaction management, and security management.

For information about how to program using the JCA resource adapters, see *Developing Connector Applications for CICS*, SG24-7714:

http://www.redbooks.ibm.com/abstracts/sg247714.html?Open

You can also consult the CICS TG Information Center:

http://pic.dhe.ibm.com/infocenter/cicstgmp/v9r0/index.jsp

12.3 Topologies

Figure 12-1 on page 263 describes the protocols supported for each topology.

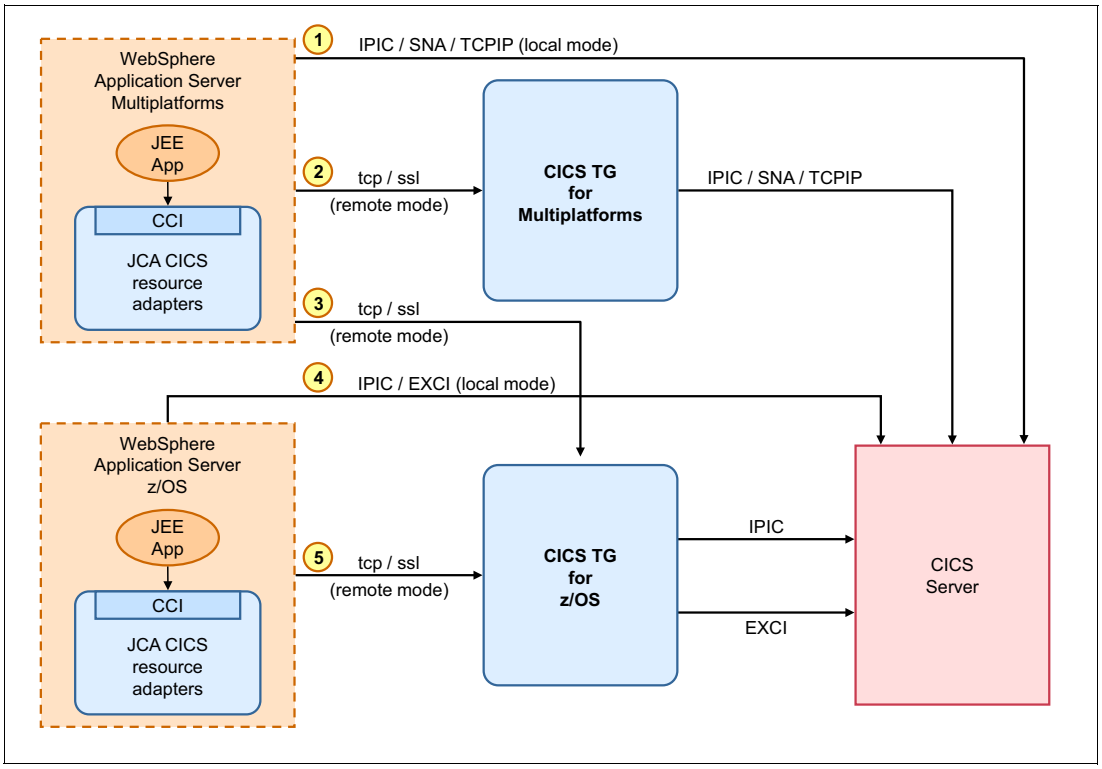


Figure 12-1 Commonly used WebSphere Application Server topologies

The following list describes supported topologies, where the topology numbers refer to the numbers used in Figure 12-1:

- WebSphere Application Server for Multiplatforms:
  - Topology 1: Local mode IPIC, SNA, or TCP/IP connections to CICS servers:
    - XA transactions are supported for IPIC connections to CICS TS.

    Note: The TCP/IP or SNA network connections from the Client daemon to the CICS region are managed and reused by the Client daemon component of the CICS TG. They are not subject to the JCA connection pooling system contract.

  - Topology 2: Two-tier or 3-tier remote mode connections to a Multiplatforms Gateway daemon, providing IPIC, SNA, or TCP/IP connections to CICS servers:

    Note: XA transactions are not supported for remote mode connections to Gateway daemons on Multiplatforms.

    TCP/IP based communication between Linux for System z and z/OS can use the IBM HiperSockets™ function to provide a highly efficient cross-memory transport. Therefore, configurations with WebSphere Application Server and CICS TG installed on Linux for System z can benefit from the HiperSockets function when using TCP/IP or IPIC connections to CICS TS.

– Topology 3: Three-tier remote mode connection to a z/OS Gateway daemon, providing external CICS interface (EXCI) or IPIC connections to CICS TS regions:
  - XA transactions are supported.

> **Note:** If CICS TG for z/OS and CICS TS are not configured in the same logical partition (LPAR), EXCI is only supported for ECI SYNCONRETURN requests.

- WebSphere Application Server for z/OS:
  – Topology 4: Local mode EXCI or IPIC connection to CICS TS regions:
    - XA transactions are supported.
  – Topology 5: Two-tier or 3-tier remote mode connection to a z/OS Gateway daemon, providing EXCI or IPIC connections to CICS TS regions:
    - XA transactions are supported.

> **Note:** If CICS TG for z/OS and CICS TS are not configured in the same LPAR, EXCI is only supported for ECI SYNCONRETURN requests.

This book focuses on the following topologies:

- WebSphere Application Server for Multiplatforms using a local mode IPIC connection to CICS TS, as shown in topology 1 in Figure 12-1 on page 263.
- WebSphere Application Server for Multiplatforms using a remote mode connection to a Gateway daemon on z/OS, which connects to CICS TS using IPIC, as shown in topology 3 in Figure 12-1 on page 263.
- WebSphere Application Server for z/OS using a local mode IPIC connection to CICS TS, as shown in topology 4 in Figure 12-1 on page 263.

Table 12-1 lists the values used in this chapter to configure WebSphere Application Server to communicate with the CICS TS region configured in Chapter 9, “Configuring IPIC in CICS Transaction Gateway for z/OS” on page 147 and the z/OS Gateway daemon configured in Chapter 10, “CICS Transaction Gateway for z/OS configuration” on page 171.

Table 12-1 Resource definition values used in this chapter

Component	Attribute	Value
CICS TS	Host name	`lp01.redbooks.ibm.com`
CICS TS	IPIC TCPIPSERVICE port	`9001`
z/OS Gateway daemon	Host name	`lp01.redbooks.ibm.com`
z/OS Gateway daemon	TCP protocol handler port	`2016`
z/OS Gateway daemon	SSL protocol handler port	`8070`
z/OS Gateway daemon	SECTION IPICSERVER	`CICSTOR1`
WebSphere Application Server	Node name	`itsoNode01`
WebSphere Application Server	Server name	`itsoServer01`
WebSphere Application Server	Cluster name	`itsoCluster`

12.4 WebSphere Application Server settings

The management of connections, transactions, and security is controlled within WebSphere Application Server through a combination of configuration parameters and application deployment descriptors. This section focuses on the configuration parameters.

12.4.1 Connection management

In WebSphere Application Server, connections for use by the ECI resource adapter are controlled through the use of a connection pool. Each Java EE Connector Architecture connection factory defines its own connection pool.

If the Java EE Connector Architecture connection factory is defined at a cluster scope, a separate instance of the connection pool is created in each application server that uses connection factory. For example, for a cluster with two application servers that use the same Java EE Connector Architecture connection factory with a connection pool Maximum Connections property setting of `100`, you can generate up to 200 connections (two servers times 100 connections).

For information about defining and customizing connection pool settings, see 12.6.2, “Configuring the connection pool properties” on page 280.

12.4.2 Transaction management

The ECI resource adapter supports two-phase commit global transactions from WebSphere Application Server for the following topologies:

- Using CICS TG for Multiplatforms, XA transactions are only supported for local mode IPIC connections to CICS TS.
- Using CICS TG for z/OS, XA transactions are supported for both local and remote mode IPIC or EXCI connections to CICS TS.

However, if you are using the ECI resource adapter for local transaction support, the scope of the transaction is limited to the Resource Manager (that is, the associated connection factory and the specified CICS server). These Resource Manager local transactions only support one-phase commit processing. WebSphere Application Server provides last-participant support to allow an ECI interaction to participate in a global transaction, provided that it is the only one-phase-commit resource in the global transaction. This approach has an increased risk that a transaction has a heuristic outcome.

The WebSphere Application Server transaction service is a server runtime component that can coordinate updates to multiple resource managers to ensure atomic updates of data. See the WebSphere Application Server Information Center for detailed information about these settings:

http://ibm.co/1ldLJIc

Table 12-2 provides a summary of the settings that are available for controlling transaction timeouts and heuristic outcomes.

Table 12-2 WebSphere Application Server Transaction service settings

Property	Description
Total transaction lifetime timeout	Default maximum time allowed for a transaction started on this server before the transaction service initiates timeout completion. Any transaction that does not begin completion processing before this timeout occurs is rolled back.
Maximum transaction timeout	Maximum time for transactions that run in this server to complete. This value needs to be greater than or equal to the total transaction lifetime timeout *and* greater than or equal to the application component timeout.
Heuristic retry limit	Number of times that the application server retries a completion signal, such as commit or rollback, after a transient exception from a resource manager or remote partner. If the application server abandons the retries, the resource manager or remote partner is responsible for ensuring that the resource or partner branch of the transaction is completed appropriately. The application server raises (on behalf of the resource or partner) an exception that indicates a heuristic hazard.
Heuristic retry wait	Time that the application server waits before retrying a completion signal, such as commit or rollback, after a transient exception from a resource manager or remote partner.
Enable logging for heuristic reporting	Enable application server transaction service to log when it is about to commit a one-phase commit resource from transactions that involve both a one-phase commit and two-phase commit resources. This is useful when you use last participant support, where there is an increased risk that a transaction has a heuristic outcome.
Heuristic completion direction	Specifies the direction that is used to complete a transaction that has a heuristic outcome; either the application server commits or rolls back the transaction, or it depends on manual completion by the administrator.

Transaction service properties are defined at the application server scope and can be configured using the following steps:

1. Log in to the administration console.
2. Click the *serverName* from the **Servers** → **Server types** → **WebSphere Application Servers** navigation menu.
3. Select **Container Services** → **Transaction service**.
4. Select the **Configuration** tab
5. From the General Properties dialog box, edit the properties, as required.
6. Click **OK**.
7. Save the changes to the master configuration.
8. If you are using a clustered WebSphere Application Server configuration, repeat for each application server in your cluster and synchronize the changes to the nodes.
9. Restart the application server or cluster for the changes to take effect.

12.4.3 Security Management

The security considerations discussed in Chapter 8, “Security” on page 131 are applicable to topologies using the ECI resource adapter with WebSphere Application Server.

The Java EE Connector Architecture connection factory custom properties include the following security-related options:

- `userName` and `password`

  The user ID and password to be sent to CICS if no other security credentials are available.

- `ClientSecurity` and `ServerSecurity`

  The fully-qualified names of the `ClientSecurity` and `ServerSecurity` classes to use in each interaction with CICS.

  For more information about the CICS TG Security classes, see the CICS TG Information Center:

  http://ibm.co/1jZe7Df

- `keyRingClass` and `keyRingPassword`

  The fully qualified name and password of the Java keystore to use on SSL connections. This stores the certificate used to identify WebSphere Application Server during client authentication on an SSL connection to CICS TS (local mode over IPIC) or CICS TG (remote mode). It also stores the public certificate expected to be received from either CICS TS or CICS TG as a trusted certificate, so that it can authenticate the remote system during the SSL handshake.

- `cipherSuites`

  Can be used when establishing an SSL connection to restrict the cipher suites that this connection factory can use.

An alternative to hardcoding `userName` and `password` in the Java EE Connector Architecture connection factory custom properties is to use either of the following options:

- Component-managed sign-on: Security credentials are propagated to CICS by the Java EE application. They are used when the res-auth element in the deployment descriptor resource reference is set to `Application`.
- Container-managed sign-on: Security credentials are propagated to CICS by the WebSphere Application Server Web or Enterprise JavaBeans (EJB) container. Used when the res-auth element in deployment descriptor resource reference is set to `Container`.

> **Note:** Container-managed security is suggested because it is a preferred practice to separate the business logic of an application from qualities of service, such as security.

To use container-managed sign-on, a Java Authentication and Authorization Service (JAAS) authentication alias is defined to WebSphere Application Server that contains the user ID and password. This user ID and password combination is then associated with a specific connection factory using the resource reference mapping option “modify resource authentication method” of the Java EE application. Applying the JAAS authentication alias to the application resource reference is more secure than applying it directly to the Java EE Connector Architecture connection factory.

When using CICS TG for z/OS, the following additional security options are available:

- *Identity propagation*, a unified security solution that enables additional user auditing and authorization by passing a distributed identity to CICS instead of a user ID and password.

  For details about configuring identity propagation, see the CICS TG Information Center:

  http://ibm.co/1szBAx5

  You can also consult *z/OS Identity Propagation*, SG24-7850:

  http://www.redbooks.ibm.com/abstracts/sg247850.html

- Client certificate mapping to associate an X.509 certificate with a RACF user ID is supported in remote mode connections to CICS TG.

  For details about configuring client certificate mapping, see the CICS TG Information Center:

  http://ibm.co/1raxcCT

  Further description of the security considerations for the topology with WebSphere Application Server on Multiplatforms using a remote mode connection to CICS TG for z/OS is available in *CICS and SOA: Architecture and Integration Choices*, SG24-5466, which is available at this website:

  http://www.redbooks.ibm.com/abstracts/sg245466.html

12.4.4 WebSphere Application Server z/OS-specific settings for EXCI

Note: This book does not cover topologies using EXCI; however, the following information is included for reference.

Extra configuration is required in WebSphere Application Server for z/OS if using local mode EXCI connections to connect to CICS TS. The following steps are required:

1. Define the EXCI libraries to WebSphere Application Server:

   Add the libraries containing any customized versions of the EXCI options table (DFHXCOPT) and the EXCI load modules to the **STEPLIB** environment variable of the WebSphere Application Server servant region start procedure. Ensure that any load library that contains a customized version of the EXCI options table DFHXCOPT is ahead of the CICS EXCI load library SDFHEXCI in the concatenation order.

2. Define a RACF surrogate security profile:

   If surrogate security checking is enabled in the EXCI options table DFHXCOPT (SURROGCHK=YES), when a Java EE server flows an EXCI request to CICS TS using CICS TG in local mode, the Java EE server will be subject to RACF surrogate security checking. The WebSphere Application Server servant region user ID requires access to act as a surrogate for the user ID flowed in the EXCI request. This can be achieved by granting the WebSphere Application Server servant region user ID read access to the *userid*.DFHEXCI SURROGAT class profile, where *userid* is the user ID flowed in the EXCI request.

   For example, to authorize the WebSphere Application Server servant region user ID WASUSER to switch to the user ID EXCIUSER, which is flowed in the ECI request, use this:

   ```
   RDEFINE SURROGAT EXCIUSER.DFHEXCI UACC(NONE) OWNER(EXCIUSER)
   PERMIT EXCIUSER.DFHEXCI CLASS(SURROGAT) ID(WASUSER) ACCESS(READ)
   ```

It is possible to disable surrogate security by either reassembling the EXCI options table DFHXCOPT, with `SURROGCHK=NO`, or by using a generic RACF surrogate profile with universal READ access, such as this example:

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

3. Set WebSphere Application Server environment variables as described in the topic *Deploying the ECI resource adapter on WebSphere Application Server for z/OS* in the CICS TG Information Center:

   – **`CTG_EXCI_INIT`**`=YES`
   – **`CTG_PIPE_REUSE`**
   – **`DFHJVPIPE`**

   Follow these steps to define WebSphere Application Server environment variables:

   a. Log in to the administration console. Then, select **Environment** → **WebSphere variables** from the navigation menu.

   b. Set the Scope as appropriate, and click **New**.

   c. Enter the Name and Value environment variables and optionally the Description variable:

      - Name: `CTG_EXCI_INIT`
      - Value: `NO`

   d. Click **OK**.

   e. Save the changes to the master configuration. If you are using a clustered WebSphere Application Server configuration, synchronize the changes to the nodes.

Note: For a local connection, the JCA connection pool that is managed by WebSphere Application Server is a set of local connection objects. The connection objects do not map onto network connections, and they do not map onto EXCI pipes because EXCI pipes are allocated directly by the Java virtual machine (JVM) threads within the Java EE servant region.

12.5 Installing the ECI resource adapter

This section describes how to install the ECI resource adapter using the WebSphere Application Server administration console. The installation of an ECI resource adapter is similar for all the topologies shown in Figure 12-1 on page 263. The instructions indicate the differences that are required for local and remote mode topologies.

The following list notes important prerequisites:

- WebSphere Application Server is installed and configured with either a stand-alone server or application server cluster, and the administration console and administration user ID are available.
- A compatible version of the ECI resource adapter file (`cicseci.rar`) is available on either a machine with browser access to the WebSphere Application Server administration console (local file system) or on the same machine as the WebSphere Application Server installation, the file must be readable by the WebSphere Application Server administration user ID (remote file system):
  - For information about selecting a suitable version of the CICS resource adapter, see the technote *"Choosing the correct CICS Resource Adapter to deploy in WebSphere Application Server"* at the following website:

    http://www.ibm.com/support/docview.wss?uid=swg21306754
  - The `cicseci.rar` file is in the CICS TG <*install_path*>/`deployable` directory.
  - The `cicseci.rar` files for previous versions of CICS TG are available from SupportPac CC03: CICS JCA resource adapters for use with Java EE application servers at the following website:

    http://www.ibm.com/support/docview.wss?uid=swg24008817

If you want to upgrade from a previous version of the ECI resource adapter, see 12.5.2, "Upgrading the ECI resource adapter" on page 273.

12.5.1 Install the ECI resource adapter

Follow these steps to install the ECI resource adapter into WebSphere Application Server:

1. Log in to the WebSphere Application Server administration console.
2. Select **Resource Adapters** from the **Resources** → **Resource Adapters** section of the navigation menu.
3. Click **Install RAR**.
4. From the Install RAR file dialog box that is shown in Figure 12-2 on page 271, complete these steps:
   a. Set the Scope to the WebSphere Application Server node where the resource adapter is to be installed to, for example, `itsoNode01`.

> **Important:** Resource adapters can only be installed at the node scope. If you are installing to an application server cluster, the resource adapter must be installed into each of the nodes used by the cluster.

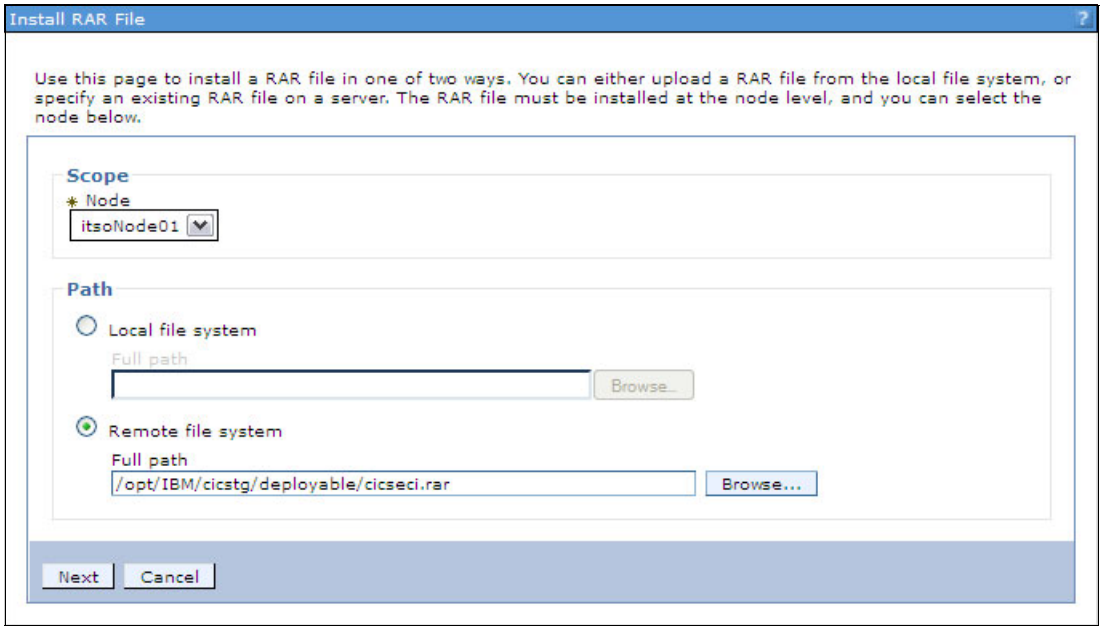


Figure 12-2 Install RAR File dialog box

b. Specify the installation path for the RAR file in the Path section:

- Select **Local file system** if the `cicseci.rar` file is on the system from which you are accessing the browser.
- Select **Remote file system** if the `cicseci.rar` file is on the same system as WebSphere Application Server.
- Click **Browse** to navigate the file system, select the `cicseci.rar` file, and click **Open**.

c. Click **Next**.

5. The Resource adapters General Properties dialog box shown in Figure 12-3 on page 272 includes a predefined name and description for the ECI resource adapter:

a. Accept the default name and description.

b. If WebSphere Application Server is using a local mode connection to CICS TG, set the Native library path to the CICS TG `bin` directory, for example, `/opt/IBM/cicstg/bin`.

> **Native library path:** The Native library path value is not required if using CICS TG in remote mode.

c. Click **OK**.

Figure 12-3 on page 272 shows the Resource adapters General Properties dialog box.

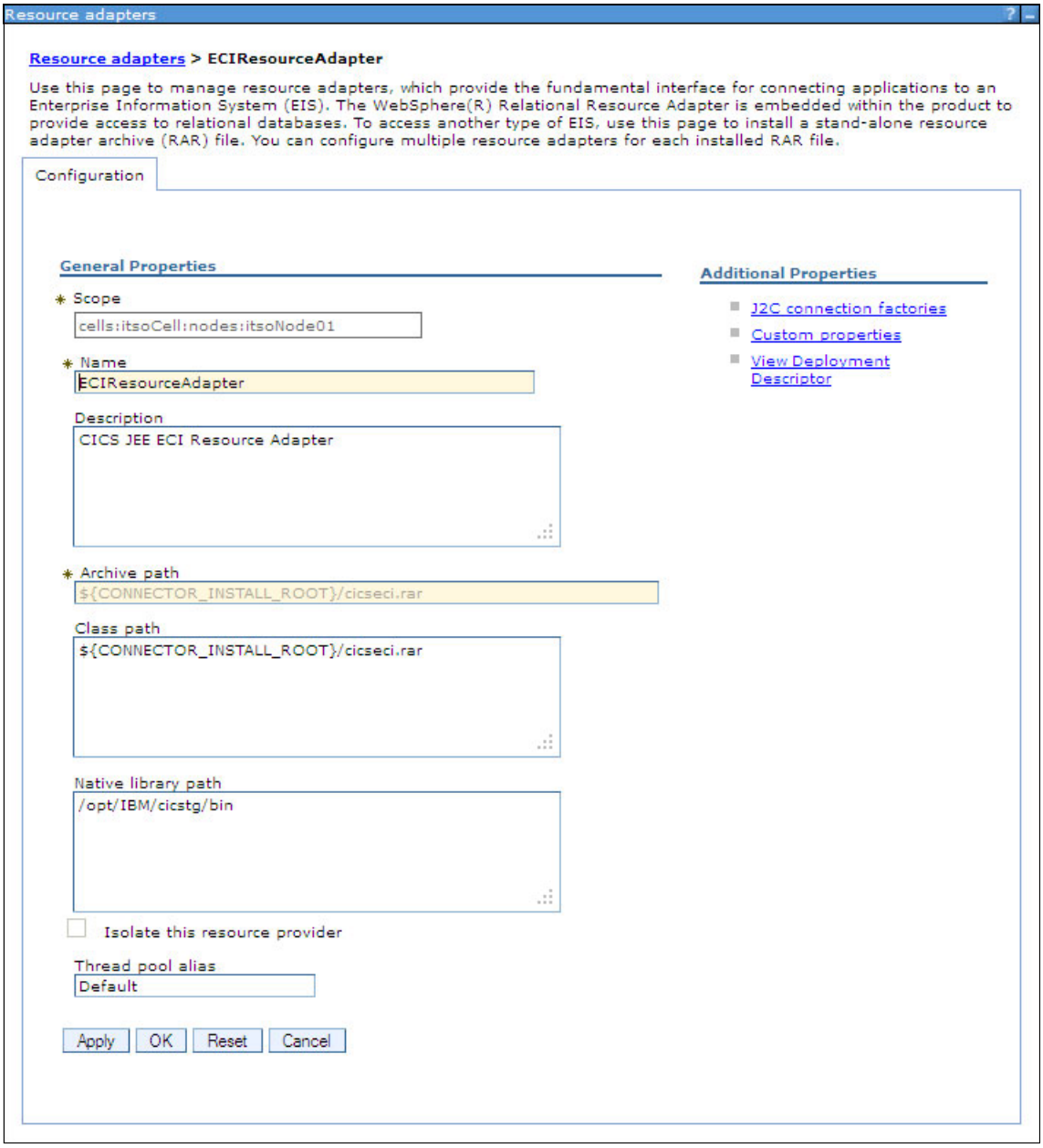


Figure 12-3 Resource adapter General Properties dialog box

6. The administration console returns you to the Resource adapters dialog box and prompts you to save the changes:
   a. Set the Scope to `All scopes`. The new resource adapter `ECIResourceAdapter` is now visible in the table of available resource adapters.
   b. Save the changes to the master configuration.
7. If you are using a WebSphere Application Server cluster configuration, repeat the previous steps to install the ECI resource adapter to all nodes in the cluster. This extra configuration is required so that the `cicseci.rar` file is available to all application servers in the cluster at run time.

8. If you are using a WebSphere Application Server cluster configuration, create a resource adapter instance at the cluster scope.

   Important: Java EE Connector Architecture connection factories can be created only at the same scope as the resource adapter that is installed. To define one Java EE Connector Architecture connection factory for use by a cluster, you must first define a resource adapter reference at the cluster scope.

   Use the following steps:

   a. Select **Resource Adapters** from the **Resources** → **Resource Adapters** section of the navigation menu.
   b. Set the Scope to the cluster, for example, `itsoCluster`.
   c. Click **New**. Then, enter `ECIResourceAdapterCluster` for Name. Optionally, enter a description, for example, `ECI Resource Adapter for cluster`.
   d. In Archive Path, select **Choose an archive path from the list of installed RARs (recommended)**. This path defaults to the `${CONNECTOR_INSTALL_ROOT}/cicseci.rar` file if the resource adapter is installed to all nodes in the cluster.
   e. If WebSphere Application Server is using a local mode connection to CICS TG, set the Native library path to the CICS TG `bin` directory, for example, `/opt/IBM/cicstg/bin`.

      Native library path: The Native library path value is not required if using CICS TG in remote mode.

   f. Click **OK**. Save the changes to the master configuration and synchronize the changes to the nodes.

For instructions to create a Java EE Connector Architecture connection factory definition, see 12.6, “Creating and configuring a Java EE Connector Architecture connection factory” on page 276.

12.5.2 Upgrading the ECI resource adapter

If you have an existing version of the ECI resource adapter installed, you can upgrade to a more recent version resource adapter, if the versions are compatible and the resource adapters implement the same version of the JCA specification. For CICS TG, for example, you can perform these tasks:

- Upgrade from CICS TG V7.2 to the V8.0 resource adapter, because they both implement the JCA 1.5 specification.
- Upgrade from CICS TG V8.1 to the V9.0 resource adapter, because they both implement the JCA 1.6 specification.
- *Do not* upgrade from CICS TG V7.2 or V8.0 to the V8.1 or V9.0 resource adapter, because of differences between the JCA 1.5 and 1.6 specifications.

An attempt to upgrade to an incompatible version resource adapter, for example, upgrading from CICS TG Version 8.0 to the 9.0 resource adapter will be rejected with the message:

```
Error Incompatibilities were found when comparing the new RAR with the selected
resource adapter or resource adapters. The following incompatibilities were found:
Error J2CA0660I: The new RAR file is not compatible with the Resource Adapter. The
JCA Spec versions of the Resource Adapter and RAR are different. Resource Adapter
JCA version: 1.5. RAR File JCA Version: 1.6.
```

Note: The Update RAR function was introduced in WebSphere Application Server Version 7.0, which supports JCA 1.5. Use of JCA 1.6 resource adapters is supported from WebSphere Application Server Version 8.0.

The advantage of upgrading a resource adapter is that the associated Java EE Connector Architecture connection factory definitions are retained. Also, any new custom properties introduced by the newer version resource adapter are added to the existing Java EE Connector Architecture connection factory definitions, with the option to either accept the default value or customize the value as required on a per connection factory basis.

Follow these steps to check which version resource adapter is currently installed:

1. Log in to the WebSphere Application Server administration console. Then, select **Resource Adapters** from the **Resources** → **Resource Adapters** section of the navigation menu.
2. Click the resource adapter name, for example, `ECIResourceAdapter`. Then, select **View Deployment Descriptor** from Additional properties. The resourceAdapter-version element displays the value, for example, for CICS TG Version 9.0.0.1, `<resourceadapter-version> 9.0.0.1</resourceadapter-version>`.

Follow these steps to upgrade the ECI resource adapter in WebSphere Application Server:

1. First, back up your existing WebSphere Application Server configuration by using the **`backupConfig`** command. See the WebSphere Application Server Information Center for details:

   http://ibm.co/1kEqgbehttp://ibm.co/1kEqgbe
2. Log in to the WebSphere Application Server administration console. Then, select **Resource Adapters** from the **Resources** → **Resource Adapters** section of the navigation menu.
3. From the Resource adapters dialog box, set the Scope to `All scopes`. Then, select the resource adapter to update, and click **Update RAR**.
4. From the Update a resource adapter dialog box, specify the installation path for the new RAR file, in the Path:
   - Select **Local file system** if the `cicseci.rar` file is on the system from which you are accessing the browser.
   - Select **Remote file system** if the `cicseci.rar` file is on the same system as WebSphere Application Server.
5. Click **Browse** to navigate to the file system, select the `cicseci.rar` file, and click **Open**. Then, click **Next**.
6. Review the configuration information that is displayed for the new RAR file.

   The information displayed includes Name, Current RAR version, New RAR version, Scope, and any resource adapter instances created from the installed resource adapter, which are marked with an asterisk (`*`). The resource adapter instances will be automatically updated at the same time.

Figure 12-4 shows an example when upgrading the `cicseci.rar` from CICS TG Version 8.1.0.1 to Version 9.0.0.1 in a configuration where a resource adapter instance `ECIResourceAdapterCluster` was previously created from the resource adapter `ECIResourceAdapter` at the cluster scope.

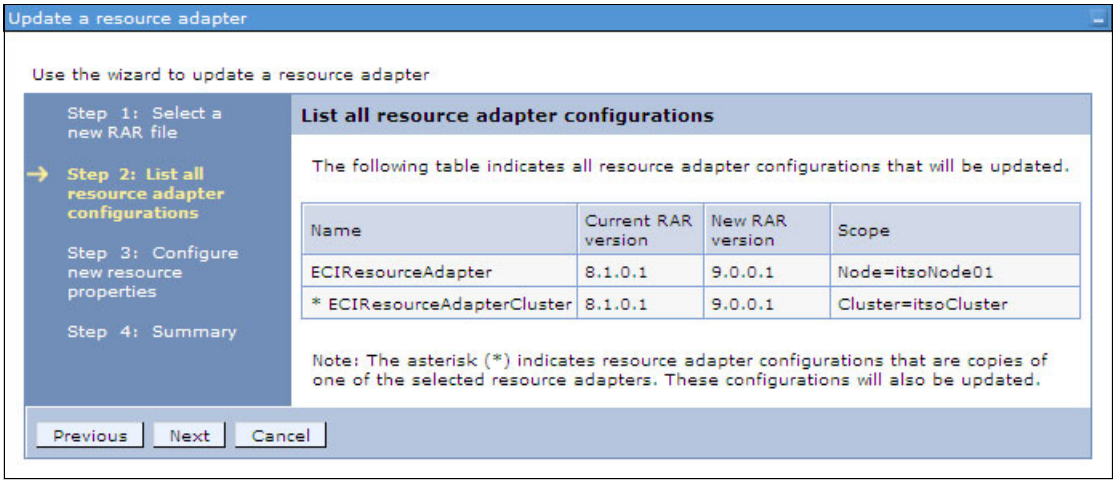


Figure 12-4 List of resource adapter configurations to be upgraded

7. Click **Next**.
8. From the Configure new resource properties dialog box as shown in Figure 12-5 on page 276, optionally edit any properties added by the new version of the resource adapter:

> **Note:** Properties are only displayed if the new resource adapter provides properties that did not exist in the previously installed resource adapter version. You can also edit these properties in the Java EE Connector Architecture connection factory definitions after completing the update.

   a. Select a Java EE Connector Architecture connection factory name, for example, **ECI**, which is displayed under the resource adapter name in the Select a resource list.
   b. In the Edit resource properties section, use the provided table to set the values for new properties that will be applied to the selected resource (the default values are displayed).
   c. Optional: Select the **Set for all** check box to apply the property value to all the resources of the same type.
   d. Click **Apply**.
   e. Click **Next**.

   Figure 12-5 on page 276 shows an example of the Configure new resource properties dialog box when upgrading from the CICS TG Version 8.1.0.1 to Version 9.0.0.1 resource adapter. This shows the introduction of a new custom property `ipicSendSessions`.

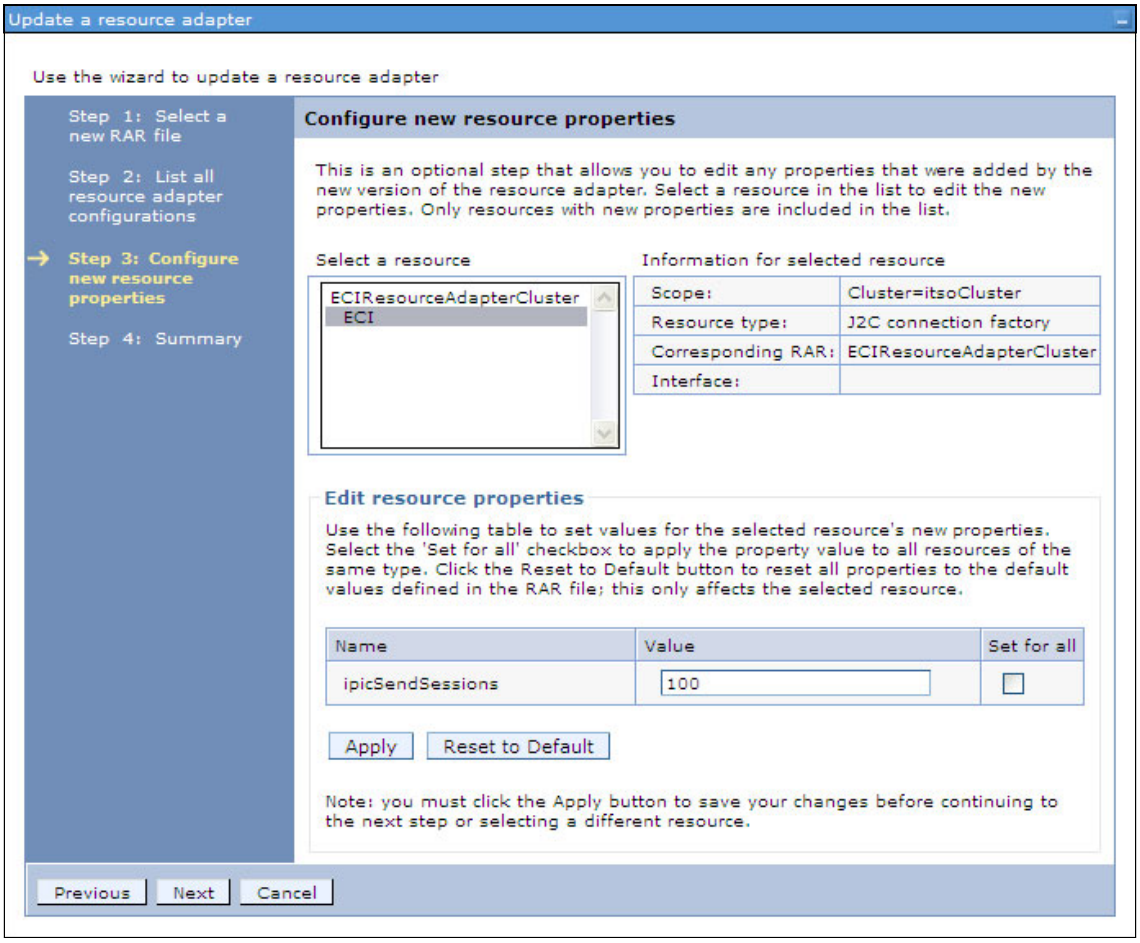


Figure 12-5 Configure new resource properties dialog box

9. Review the summary panel and click **Finish**.

   When you click Finish, all the configuration changes are saved automatically.

10. Restart the application servers that contain the updated RAR file.

For instructions to create a Java EE Connector Architecture connection factory definition, see 12.6, “Creating and configuring a Java EE Connector Architecture connection factory” on page 276.

12.6 Creating and configuring a Java EE Connector Architecture connection factory

This section describes how to use the WebSphere Application Server administration console to create and configure a Java EE Connector Architecture connection factory for the ECI resource adapter. It covers both local and remote mode IPIC connections to CICS TS. The creation of a Java EE Connector Architecture connection factory is similar for all the topologies shown in Figure 12-1 on page 263. The following instructions indicate the differences that are required for local and remote mode topologies.

A basic prerequisite to getting started is installing the ECI resource adapter; see 12.5.1, “Install the ECI resource adapter” on page 270.

12.6.1 Creating a Java EE Connector Architecture connection factory

This section describes how to create a Java EE Connector Architecture connection factory for the ECI resource adapter using the WebSphere Application Server administration console.

Use the following steps to create a Java EE Connector Architecture connection factory in WebSphere Application Server:

1. Log in to the WebSphere Application Server administration console.
2. Select **Resource Adapters** from the **Resources** → **Resource Adapters** section of the navigation menu.
3. Set the Scope:
   - If you are using a stand-alone WebSphere Application Server configuration, select the node Scope to which you installed the resource adapter, for example, `itsoNode01`.
   - If you are using a clustered WebSphere Application Server configuration, select the cluster Scope to which you defined the resource adapter instance, for example, `itsoCluster`.
4. Click the name of the resource adapter or resource adapter instance. This ensures that the Java EE Connector Architecture connection factory is associated with the correct resource adapter:
   - If you are using a stand-alone WebSphere Application Server configuration, select the resource adapter, `ECIResourceAdapter`.
   - If you are using a clustered WebSphere Application Server cluster configuration, select the resource adapter instance, `ECIResourceAdapterCluster`.
5. From the Additional properties section, click **J2C connection factories**.
6. Click **New** to create a new Java EE Connector Architecture connection factory using the ECI resource adapter.
7. The Java EE Connector Architecture connection factories General Properties dialog box is displayed, as shown in Figure 12-6 on page 279. This allows you to define the name, Java Naming and Directory Interface (JNDI) name, and description for the Java EE Connector Architecture connection factory. Each Java EE Connector Architecture connection factory must have a unique Name and JNDI name.
   - The Name is used by WebSphere Application Server administration to reference this Java EE Connector Architecture connection factory; it is not used by Java EE applications. It must be unique within the WebSphere Application Server cell.
   - The JNDI name is used by Java EE applications to access the Java EE Connector Architecture connection factory. It provides a single symbolic name for applications to access CICS and masks the details of the underlying Gateway daemon and CICS topology. A common naming convention is to prefix the JNDI name with `eis` to indicate that it is a JNDI name for an enterprise information system.

   a. Enter the Name:
      - For the local mode topologies (numbers 1 and 4 in Figure 12-1 on page 263), enter `ECILocal` for Name.
      - For the remote mode topology (number 3 in Figure 12-1 on page 263), enter `ECIRemotezOS` for Name.

b. Enter the JNDI Name:

- For the local mode topologies (numbers 1 and 4 in Figure 12-1 on page 263), enter eis/`ECILocal` for JNDI Name.
- For the remote mode topology (number 3 in Figure 12-1 on page 263), enter eis/`ECIRemotezOS` for JNDI Name.

c. Enter a Description. The description is optional but it can help to identify the purpose of the connection factory.

For the local mode topologies (numbers 1 and 4 in Figure 12-1 on page 263), you can enter the following description:

```
Enter Local mode IPIC connection to CICS TS for Description
```

For the remote mode topology (number 3 in Figure 12-1 on page 263), you can enter the following description:

```
Enter Remote mode connection to CICS TG for z/OS for Description
```

Configuration note: The Java EE Connector Architecture connection factory General properties dialog box also includes security settings for authentication that you can configure. This example does not flow authentication credentials and accepts the default values.

d. Click **OK**.

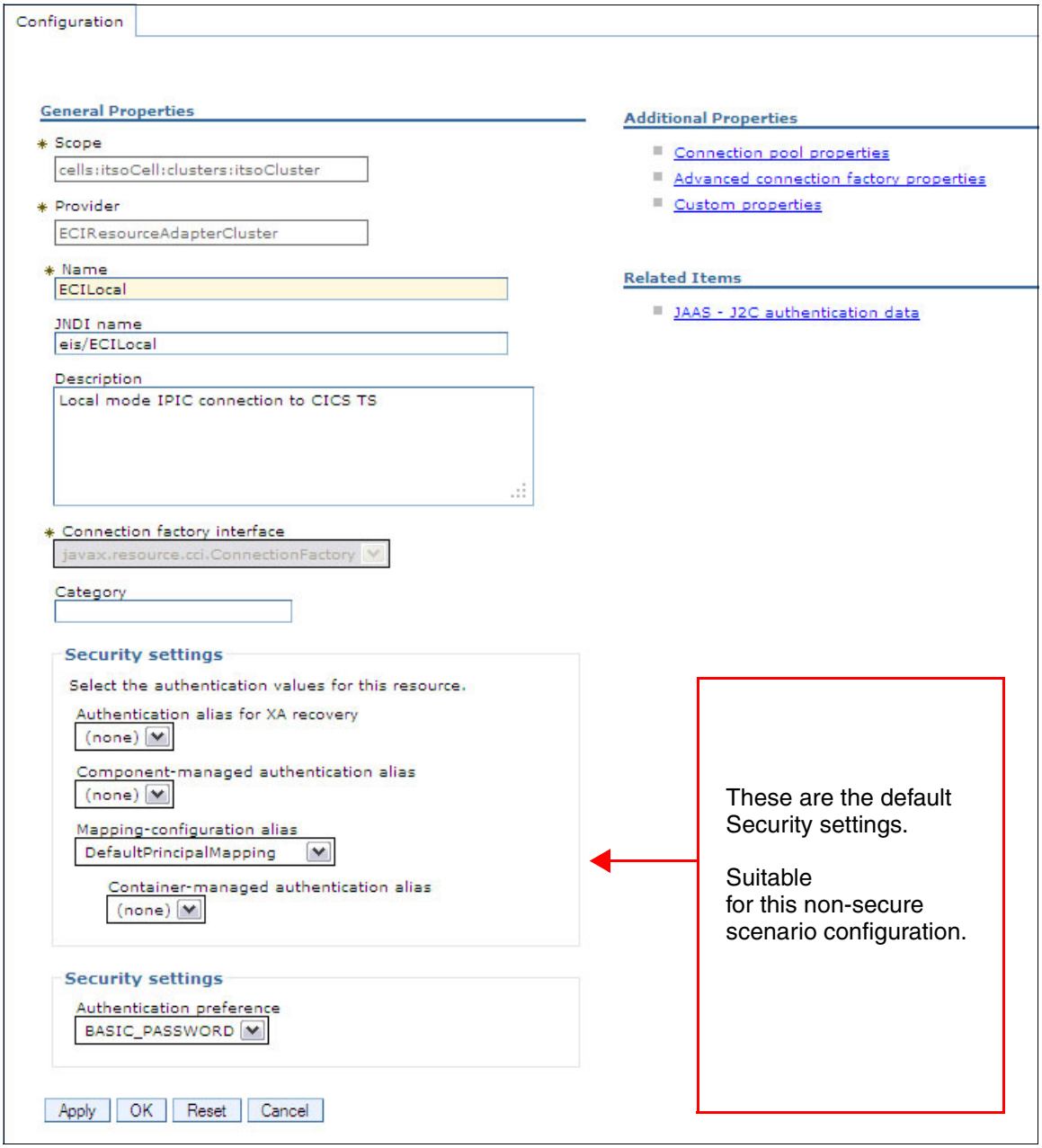


Figure 12-6 Java EE Connector Architecture connection factory general properties for the local mode topology

8. The administration console returns you to the Java EE Connector Architecture connection factories dialog box. The new Java EE Connector Architecture connection factory is now visible in the table:
   a. Save the changes to the master configuration.
   b. If you are using a clustered WebSphere Application Server configuration, synchronize the changes to the nodes.

12.6.2 Configuring the connection pool properties

This section describes how to configure the connection pool properties for a Java EE Connector Architecture connection factory. In remote mode, the connection pool represents physical network connections between the WebSphere Application Server and the Gateway daemon.

Table 12-3 describes the connection pool properties available on a Java EE Connector Architecture connection factory.

Table 12-3 Java EE Connector Architecture connection factory connection pool properties

Property	Description
`Connection timeout`	Interval after which a connection request times out and a `ConnectionWaitTimeoutException` is thrown. Occurs when there are no connections available in the free pool and no new connections can be created. This usually occurs because the maximum connections value has been reached. Consider reducing this value to be sufficiently long enough to queue requests but to time out requests early if no connection becomes available.
`Maximum connections`	Maximum number of physical connections that can be created in this pool. These are the physical connections to the back-end resource. This needs to take into account the number of concurrent connections requested expected during a peak load. For remote mode connections, this needs to correspond to the CICS TG maximum number of connection manager threads (`maxconnect`).
`Minimum connections`	Minimum number of physical connections to maintain. If the size of the connection pool is at or below the minimum connection pool size, the `Unused timeout` thread does not discard physical connections. However, the pool does not create connections solely to ensure that the minimum connection pool size is maintained. For remote mode connections, this needs to correspond to the CICS TG initial number of connection manager threads (`initconnect`). Consider increasing from the default value to prevent timeouts and allow smooth transition from periods of low activity to peak workloads.
`Reap time`	Interval between runs of the pool maintenance thread. Set the `Reap Time` value less than the values of `Unused timeout` and `Aged timeout`. The pool maintenance thread discards any connections remaining unused for longer than the `Unused timeout` value or that remain active longer than the `Aged timeout` value, until it reaches the number of connections specified in `Minimum Connections`. The `Reap Time` interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

Property	Description
`Unused timeout`	Interval after which unused or idle connections are discarded. Set the `Unused timeout` value higher than the `Reap timeout` value for optimal performance. Consider reducing this value so that connections are closed after a shorter period of inactivity. If TCP/IP connection balancing is in use across a group of cloned Gateway daemons, repeated failures of requests to a Gateway daemon will cause it to be removed from the workload distribution. When the Gateway daemon recovers, no connections will be established to it, if all existing connections from the client are being reused from a connection pool. To ensure that such Gateway daemons can be efficiently used in a failback situation, set `Unused timeout` to ensure that connections are recycled periodically.
`Aged timeout`	Interval before a physical connection is discarded. Set the `Aged timeout` value higher than the `Reap timeout` value for optimal performance. See the comment, which is also applicable to this parameter, under `Unused timeout` regarding Gateway daemon failback situations.
`Purge Policy`	Specifies how to purge connections when a stale connection or fatal connection error is detected. Two values are supported: ► `EntirePool`: all connections in the pool are marked stale. A connection in use is closed and issues a stale connection exception during the next operation on that connection. Subsequent getConnection() requests from the application result in new connections to the database opening. When using this purge policy, there is a slight possibility that some connections in the pool are closed unnecessarily when they are not stale. However, this closure is a rare occurrence. In most cases, a purge policy of `EntirePool` is the best choice. ► `Failing Connection Only`: Only the connection that caused the stale connection exception is closed. Although this eliminates the possibility that valid connections are closed unnecessarily, it makes recovery from an application perspective more complicated because there is a possibility that the next getConnection() request from the application can return a connection from the pool that is also stale, resulting in more stale connection exceptions.

Use the following steps to configure the connection pool:

1. Configure the connection pool properties:
   a. Click the name of the Java EE Connector Architecture connection factory, for example, `ECI`.
   b. Click **Connection pool properties** in Advanced properties. The Connection pool dialog box is displayed as shown in Figure 12-7 on page 282.
   c. Edit the properties to meet your runtime requirements. For a description of the properties, see “Connection management” on page 265.

For the purpose of this chapter, which is running the `IVT` application with small numbers of concurrent requests, we used the default values. For information about the values used in the high availability scenario, see 13.6.3, “Configuring a Java EE Connector Architecture connection factory for CICS TG remote mode” on page 317.

- Connection timeout = `180 secs`
- Maximum connections = `10`
- Minimum connections = `1`
- Reap time = `180 secs`
- Unused timeout = `1800 secs`
- Aged timeout = `0`
- PurgePolicy = `EntirePool`

Figure 12-7 shows the Connection pools dialog box.

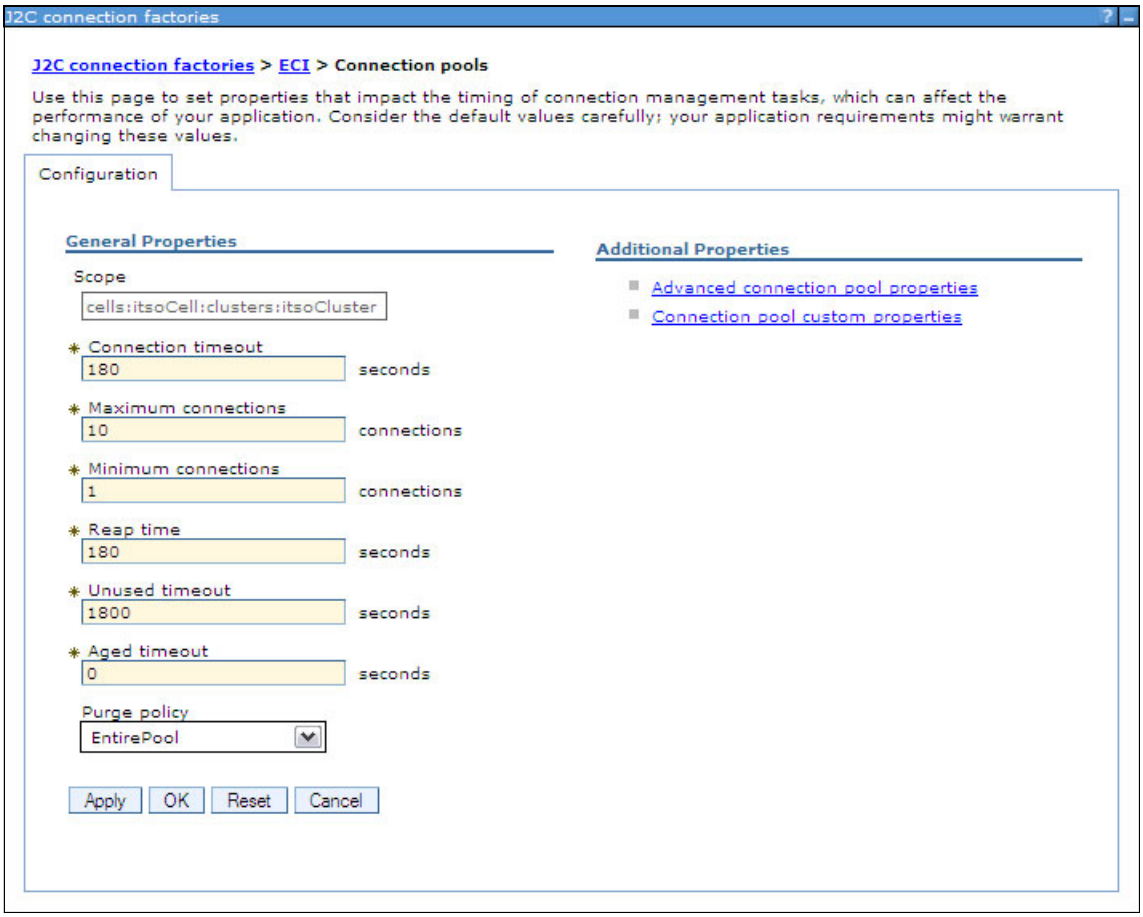


Figure 12-7 Java EE Connector Architecture connection factory connection pool default values

d. Save the changes to the master configuration.

e. If you are using a clustered WebSphere Application Server configuration, synchronize the changes to the nodes.

Before the Java EE Connector Architecture connection factory can be used, the custom properties must be configured to connect to CICS TS. This configuration is specific to the topology that you are implementing. For instructions to customize the Java EE Connector Architecture connection factory definition to connect to CICS TS, see the appropriate following section:

- 12.6.3, “Configuring a Java EE Connector Architecture connection factory for CICS TG local mode” on page 283
- 12.6.4, “Configuring a Java EE Connector Architecture connection factory for CICS TG remote mode” on page 285

12.6.3 Configuring a Java EE Connector Architecture connection factory for CICS TG local mode

This section describes how to configure the Java EE Connector Architecture connection factory custom properties for the topologies where WebSphere Application Server uses a local mode connection.

The following list notes a few examples:

- WebSphere Application Server on Multiplatforms using CICS TG for Multiplatforms in local mode to connect to CICS TS (as shown by topology 1 in Figure 12-1 on page 263)
- WebSphere Application Server on z/OS using CICS TG for z/OS in local mode to connect to CICS TS (as shown by topology 4 in Figure 12-1 on page 263)

A prerequisite to configuring a Java EE Connector Architecture connection factory is that a Java EE Connector Architecture connection factory was defined. For instructions, see 12.6, “Creating and configuring a Java EE Connector Architecture connection factory” on page 276. Then, proceed to the configuration process.

Use the following steps to configure the Java EE Connector Architecture connection factory custom properties using the WebSphere Application Server administration console:

1. Log in to the WebSphere Application Server administration console.
2. Select **J2C connection factories** from the **Resources** → **Resource Adapters** section of the navigation menu.
3. In the Java EE Connector Architecture connection factories dialog box, follow these steps:
   a. Set the Scope to `All scopes`.
   b. Select the Java EE Connector Architecture connection factory created in 12.6.1, “Creating a Java EE Connector Architecture connection factory” on page 277, for example, `ECILocal`.
4. The administration console displays the Java EE Connector Architecture connection factories General Properties dialog box. In the Additional properties section, select **Custom properties**.

5. In the Java EE Connector Architecture connection factories Custom properties dialog box, follow these steps:

   a. Edit the properties:

      - Set APPLID to `ITSOAIX`

        This is used to identify the client on the IPIC request made to CICS TS. The CICS TS server uses this information to determine the IPIC connection that will be used by the request. For example, the CICS TS default IPIC connection autoinstall program uses the last four characters to generate the IPCONN resource definition name, which must be unique for IPIC connections to that CICS TS region. It will be used as the APPLID in the installed IPCONN resource definition.

      - Set APPLIDQUALIFIER to `CTGWASA`

        This is used to identify the client on the IPIC request made to CICS TS. The CICS TS server uses this information to determine the IPIC connection that will be used by the request. It will be used as the NETWORKID in the installed IPCONN resource definition.

> **Note:** If you are using a WebSphere Application Server cluster topology and the CICS TS sample IPIC connection autoinstall program to generate unique IPCONN resource definitions, additional restrictions exist on the APPLID and APPLIDQUALIFIER values:
>
> - APPLID must be fewer than eight characters to allow CICS to append the additional unique characters.
> - APPLIDQUALIFIER must be unique for each WebSphere Application Server cluster or stand-alone application server connecting to the same CICS TS region.
>
> For more details, see “Avoiding problems using local mode IPIC connections” on page 318.

      - Set the connectionURL to `local:`

        To use a local mode connection.

      - Set serverName to `tcp://lp01.redbooks.ibm.com:9001`

        This is the IPIC URL of the target CICS server. This example uses a TCP connection to a CICS TS server with the `lp01.redbooks.ibm.com` host name whose IPIC TCPIPSERVICE is listening on port `9001`.

        Optionally, the URL can also specify the Applid or ApplidQualifier and Applid values of the target CICS region to validate that the connection is established to the intended CICS region. When the connection is established, CICS validates these values, and if they do not match the values defined for the CICS server that received the request, the connection is rejected. To do this, use the following syntax:

        ```
        protocol://hostname:port#cicsApplid
        protocol://hostname:port#cicsApplidQualifier.cicsApplid
        ```

- Set ipicSendSessions to `100`

  This is only applicable to local mode and sets the maximum number of simultaneous transactions, or CICS tasks, that is allowed over the connection. The actual number of send sessions used is determined by this connection factory property, or the **`IPCONN RECEIVECOUNT`** parameter in CICS TS, whichever is lower, negotiated at connection establishment.

  Note: The `ipicSendSessions` property was introduced in CICS TG Version 9.0.

- Accept the default values for other properties.

b. Save the changes to the master configuration.

c. If you are using a clustered WebSphere Application Server configuration, synchronize the changes to the nodes.

12.6.4 Configuring a Java EE Connector Architecture connection factory for CICS TG remote mode

This section describes how to configure the Java EE Connector Architecture connection factory custom properties for the topologies where WebSphere Application Server connects to CICS TG in remote mode, for example:

- WebSphere Application Server on Multiplatforms using CICS TG for z/OS in remote mode to connect to CICS TS (as shown by topology 3 in Figure 12-1 on page 263)

A prerequisite to configuring a Java EE Connector Architecture connection factory is that a Java EE Connector Architecture connection factory was defined. For instructions, see 12.6, “Creating and configuring a Java EE Connector Architecture connection factory” on page 276. Then, proceed to the configuration process.

Use the following steps to configure the Java EE Connector Architecture connection factory custom properties using the WebSphere Application Server administration console:

1. Log in to the WebSphere Application Server administration console.
2. Select **J2C connection factories** from the **Resources** → **Resource Adapters** section of the navigation menu.
3. In the Java EE Connector Architecture connection factories dialog box, follow these steps:
   a. Set the Scope to `All scopes`.
   b. Select the Java EE Connector Architecture connection factory that you created in the previous sections, for example, `ECIRemotezOS`.
4. The administration console displays the Java EE Connector Architecture connection factories General Properties dialog box. In the Additional properties section, select **Custom properties**.
5. In the Java EE Connector Architecture connection factories Custom properties dialog box, edit the following properties:
   – Accept the default (blank) values for `applid` and `applidqualifier`.

     These values are optional values to identify WebSphere Application Server as the client to the Gateway daemon.

   – Set `connectionURL` to `tcp://lp01.redbooks.ibm.com`.

     This URL is the URL to connect to the TCP protocol handler of the z/OS Gateway daemon on host name `lp01.redbooks.ibm.com`.

– Set `portNumber` to `2016`.

  This setting is the port of the TCP protocol handler of the z/OS Gateway daemon.

– Set `serverName` to `CICSTOR1`.

  This setting is the name of the target CICS server that is defined in an `IPICSERVER` SECTION of the CICS TG configuration file.

– Accept the default values for other properties.

6. Save the changes to the master configuration. If you are using a clustered WebSphere Application Server configuration, synchronize the changes to the nodes.

12.7 Testing end-to-end

You can use the CICS TG JCA resource adapter installation verification test (`IVT`) to test an end-to-end flow from a Java EE application running in WebSphere Application Server that is flowing an ECI request to CICS TS.

The `IVT` is a Java EE application that provides a web interface and that makes both non-transactional and transactional ECI calls to CICS using a Java EE Connector Architecture connection factory that is associated with the ECI resource adapter.

The following scenarios, as illustrated in Figure 12-1 on page 263, were tested during the writing of this book:

- Topology 1

  WebSphere Application Server on Multiplatforms using a local mode IPIC connection to CICS TS

- Topology 3

  WebSphere Application Server on Multiplatforms using a remote mode connection to CICS TG for z/OS to connect to CICS TS using IPIC

- Topology 4

  WebSphere Application Server on z/OS using a local mode IPIC connection to CICS TS

Each test is repeated using both local and two-phase commit transactions, that is, Java EE Connector Architecture connection factory custom properties, `xaSupport=off` and `xaSupport=on`.

The following prerequisites are needed to complete the test:

1. Install the ECI resource adapter (see 12.5.1, “Install the ECI resource adapter” on page 270).
2. Create a Java EE Connector Architecture connection factory (see 12.6.1, “Creating a Java EE Connector Architecture connection factory” on page 277), and configure it for local mode, for example, `ECILocal` (see 12.6.3, “Configuring a Java EE Connector Architecture connection factory for CICS TG local mode” on page 283).
3. Create a Java EE Connector Architecture connection factory (see 12.6.1, “Creating a Java EE Connector Architecture connection factory” on page 277), and configure it for remote mode, for example, `ECIRemotezOS` (see 12.6.4, “Configuring a Java EE Connector Architecture connection factory for CICS TG remote mode” on page 285).

These high-level steps are used to run the test:

1. Deploy and configure the `IVT` to use the appropriate Java EE Connector Architecture connection factory (see 12.7.1, “Deploying and configuring the IVT” on page 287).

   If the `IVT` is already installed, map the Resource Reference to the Java EE Connector Architecture connection factory (see “Mapping the resource reference to the Java EE Connector Architecture connection factory” on page 289).

2. Run the `IVT` (see 12.7.2, “Running the IVT” on page 289).

12.7.1 Deploying and configuring the IVT

Follow these steps to deploy and configure the `IVT` using the WebSphere Application Server administration console:

1. Log in to the WebSphere Application Server administration console.
2. Select **WebSphere enterprise applications** from the **Applications** → **Application type** section of the navigation menu.
3. Click **Install**.
4. From the Path to the new application dialog box, shown in Figure 12-8, specify the path of the `IVT` ear file.

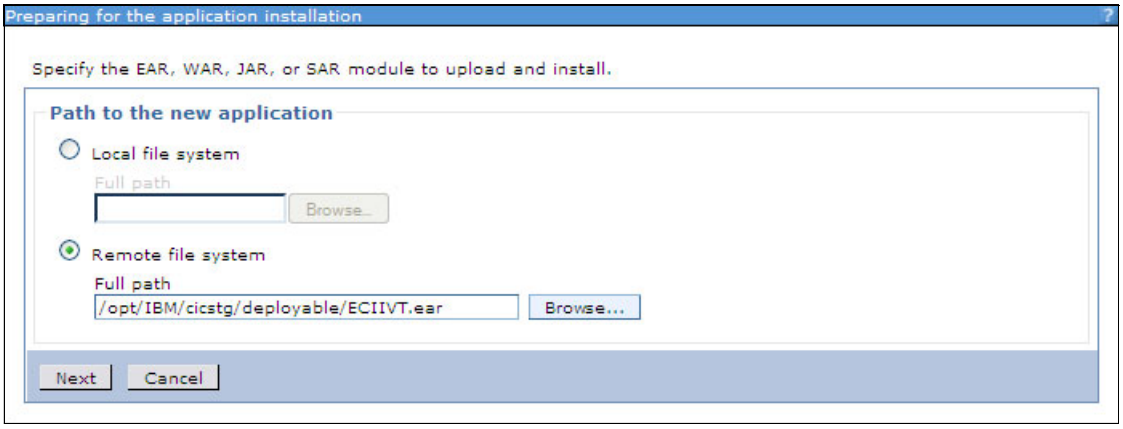


Figure 12-8 Path to the new application dialog box

   The `IVT` is provided as `ECIIVT.ear` in the `<install_path>/deployable` directory. Follow these steps:

   – Select **Local file system** if the `ECIIVT.ear` file is on the system from which you are accessing the browser.

   – Select **Remote file system** if the `ECIIVT.ear` file is on the same system as WebSphere Application Server.

   – Click **Browse** to navigate to the appropriate file system, and then select the `cicseci.rar` file. Click **Open**.

5. Click **Next**.
6. From the How do you want to install the application dialog box, shown in Figure 12-9 on page 288, select **Detailed - Show all installation options and parameters**. This option allows you to map the resource reference name that is defined in the `IVT` application with the JNDI name that you specified for the Java EE Connector Architecture connection factory.

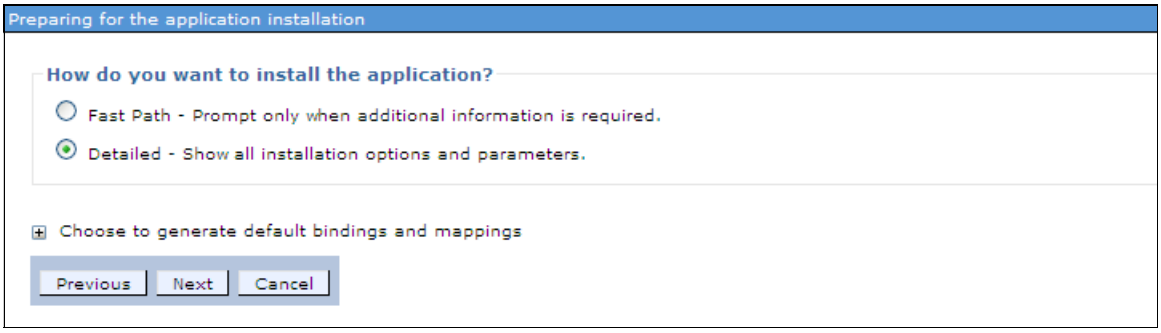


Figure 12-9 How do you want to install the application dialog box

7. Click **Next**.
8. Step through the install application wizard:
   a. Map modules to servers:

      Map the `ECIIVTEJB` and `ECIIVTWeb` modules to the WebSphere Application Server application server or cluster. If you have a separate web server, also map the `ECIIVTWeb` module to it.

   b. Map resource references to resources:

      Ensure that the Target Resource JNDI Name for the `ECIIVTEJB` module matches the JNDI name of the Java EE Connector Architecture connection factory that you defined earlier, for example, `eis/ECIRemotezOS` for the remote mode connection to the Gateway daemon on z/OS, as shown in Figure 12-10.

   c. Click **Browse** for Target Resource JNDI Name, and select the appropriate Java EE Connector Architecture connection factory from the displayed list. Then, click **Apply**.

   d. For all other steps, accept the default values. Click **Finish**.

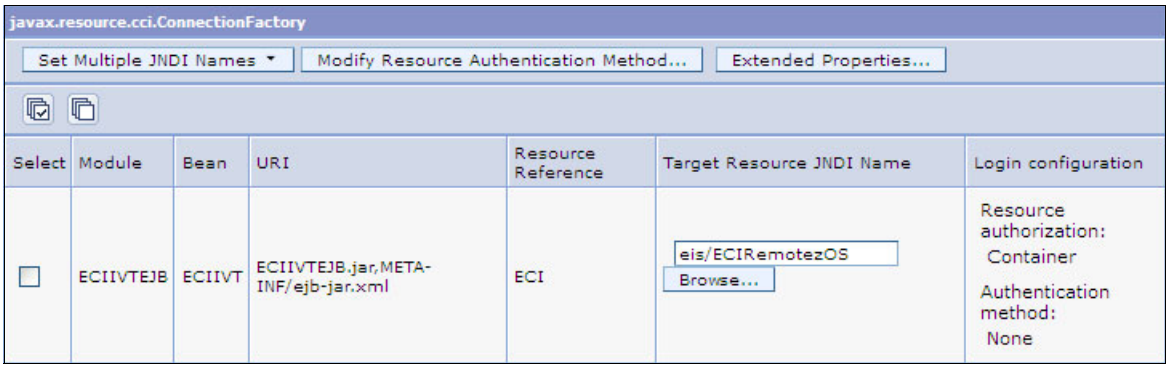


Figure 12-10 Map resource reference to resources dialog box

Wait for the installation to complete. The following message displays on the administration console:

```
Application ECIIVT installed successfully
```

9. Click **Save** to save the changes to the master configuration. If you are using a clustered WebSphere Application Server configuration, synchronize the changes to the nodes.
10. The Enterprise Application dialog box displays showing the `ECIIVT` application in a stopped state. Start the `IVT` application.

Restart note: After installing the resource adapter and defining the Java EE Connector Architecture connection factory, the WebSphere Application Server or cluster must be restarted.

Mapping the resource reference to the Java EE Connector Architecture connection factory

An alternative to specifying the deployment option is to use the “Fast Path” installation option and to update the resource reference mapping after the application is deployed. To do so, use the following steps:

1. Select **WebSphere enterprise applications** from the **Applications** → **Application type** section of the navigation menu.
2. Click the application name, for example, `ECIIVT`.
3. Click **Resource references** in the References section. Ensure that the Target Resource JNDI Name for the `ECIIVTEJB` module matches the JNDI name of the Java EE Connector Architecture connection factory as described in 12.6, “Creating and configuring a Java EE Connector Architecture connection factory” on page 276.
4. Click **OK**.
5. Save the changes to the master configuration. If the application was running, restart it to activate the change.

12.7.2 Running the IVT

Run the `IVT` by entering the following URL in a browser:

`http://`*`app_server_host`*`:port/ECIIVTWeb/index.jsp`

Here is an example:

`http://aix.redbooks.ibm.com:9080/ECIIVTWeb/index.jsp`

If the test is successful, it returns a web page that displays the date and time on the CICS server and a success confirmation message.

12.8 Configuring the XA connection

This section describes how to configure the ECI resource adapter Java EE Connector Architecture connection factory to use XA transactions. For a description of the transaction support that is provided by CICS TG when using WebSphere Application Server, see 12.4, “WebSphere Application Server settings” on page 265.

The ability to flow an XA transaction relies on using a supported topology. The ability to flow an XA transaction is controlled through a combination of configuration parameters and application deployment descriptors. The ECI resource adapter supports two-phase commit global transactions from WebSphere Application Server for the following topologies:

- Using CICS TG for Multiplatforms, XA transactions are supported only for local mode IPIC connections to CICS TS.
- Using CICS TG for z/OS, XA transactions are supported for both local and remote mode IPIC or EXCI connections to CICS TS.

The Java EE Connector Architecture connection factory for the ECI resource adapter contains a custom property `xaSupport`. To enable XA transactions, set `xaSupport=on`.

For more details about using XA transactions with WebSphere Application Server in a high availability environment, see Chapter 13, “High availability configuration” on page 295.

12.9 Flowing the user ID and password on an ECI request

If the CICS server requires user ID and password authentication for incoming requests, WebSphere Application Server needs to flow a user ID and password on the ECI request.

To use container-managed sign-on, a Java Authentication and Authorization Service (JAAS) authentication alias that contains the user ID and password is defined to WebSphere Application Server. This user ID and password combination is then associated with a specific connection using the resource reference mapping of the Java EE application using the default authentication method. It is more secure to apply the JAAS authentication alias to the application than directly to the connection factory. Follow these steps:

1. Log in to the administration console.
2. Open the Global security dialog box from the **Security** → **Global security** section of the navigation menu.
3. From the Authentication section, click **Java Authentication and Authorization Service** → **J2C authentication data**.
4. Click **New**.
5. Enter the following details for Java EE Connector Architecture authentication data:
   - Set Alias to `cicsuserAuthAlias`.
   - Set User ID to `ITSOUSER`.
   - Set Password to `password`.
   - Click **OK**.
6. Select **WebSphere enterprise applications** from the **Applications** → **Application type** section of the navigation menu.
7. Click the application name, for example, `ECIIVT`. Then, in the References section, click **Resource references**.
8. Select the module that uses the Java EE Connector Architecture connection factory, for example, `ECIIVTEJB`, and then click **Modify Resource Authentication Method**.
9. In Specify authentication method, select **Use default method**, and then select Authentication data entry from the list, for example, `scope/cicsuserAuthAlias`.

   Note: the value entered for the Java EE Connector Architecture authentication data alias is automatically prefixed with the scope at which it was created.

10. Click **Apply**.
11. Click **OK**.
12. Save the changes to the master configuration. If you are using a clustered WebSphere Application Server configuration, synchronize the changes to the nodes.

12.10 Securing the connection with Secure Sockets Layer

You can configure WebSphere Application Server for Multiplatforms to connect using SSL to both a CICS TS region using IPIC (in a local mode topology) and a z/OS Gateway daemon (in a remote mode topology). In both cases, WebSphere Application Server acts in the role of the client, and the steps that are required to configure SSL on the connection are the same.

When using the ECI resource adapter in WebSphere Application Server to establish SSL connections, a Java keystore (`.jks`) file is required to store the SSL certificates. The path to the keystore and its password are specified in the custom properties of the Java EE Connector Architecture connection factory. The keystore can be created using various key management tools, for example, the **`iKeyman`** tool. The Keystore must exist on the file system of the machine where WebSphere Application Server is installed.

To meet prerequisites, verify that the SSL is configured in the Gateway daemon and on the CICS TS IPIC connection. Obtain the SSL public certificates that are used to secure the Gateway daemon and CICS TS region from the CICS TG and CICS TS administrators. For more information, see 10.4, “Configure secure connections” on page 186 and 9.4.1, “Create a key ring for CICS” on page 163.

Configure WebSphere Application Server to use SSL on a connection to either a Gateway daemon or CICS TS region (using IPIC):

1. Create a Java keystore (`.jks`) file for use by WebSphere Application Server acting as the client end of the SSL connection:
   a. Use the **`iKeyman`** tool provided in the Java `bin` directory of the CICS TG <*install_path*>, for example, `/opt/IBM/cicstg/jvm17/bin/ikeyman`, to create a Java keystore file.

      For example, create a Java keystore file called `aix_was_key.jks` with password `redbooks` in directory `/usr/IBM/WebSphere`.
   b. Ensure that the permissions of the Java keystore file permit the user ID under which WebSphere Application Server is running to read the file.
2. For server authentication, import the SSL public certificates used to secure the Gateway daemon and CICS TS region into the WebSphere Application Server keystore as trusted certificates, which are also known as *signer certificates*.

   In the SSL configuration used in this book, the CICS TS region and z/OS Gateway daemon SSL certificates are signed with the same RACF certificate authority (CA) certificate, which was exported as `cicsroot.cer`. Use the **`iKeyman`** tool to add `cicsroot.cer` as a signer certificate to `aix_was_key.jks`.
3. If client authentication is enabled on the SSL connection in the Gateway daemon or CICS TS, follow these steps:

   > **Note:** Client authentication was not tested from WebSphere Application Server in this book.

   a. Use the **`iKeyman`** tool to create a client certificate in the WebSphere Application Server Java keystore file `aix_was_key.jks` as a personal certificate. This certificate is used by WebSphere Application Server to identify itself on the SSL connections. For example, create a certificate called `was_aix_itso`.

      > **Note:** For testing purposes, a self-signed certificate is sufficient. For production systems, the use of a third-party CA signed certificate is suggested.

b. Export the WebSphere Application Server's public certificate `was_aix_itso` and send it to the CICS TG and CICS TS administrators, so that they can add it as a trusted certificate to the RACF key rings that are used by the z/OS Gateway daemon and the CICS TS region.

4. Edit the existing Java EE Connector Architecture connection factory definition custom properties to use the SSL connections:
   - Set `keyRingClass` to the fully qualified path of the WebSphere Application Server Java keystore file, for example, `/usr/IBM/WebSphere/aix_was_key.jks`.
   - Set `keyRingPassword` to the password of the WebSphere Application Server Java keystore file, for example, `redbooks`.
   - For local mode connection to CICS TS over IPIC connection:
     - Set `serverName` to specify `ssl` as the protocol and the `CICS TS IPIC SSL` port as the port value, for example, `ssl://lp01.redbooks.ibm.com:9001`.

     Note: This assumes that the TCPIPSERVICE in CICS TS listening on port `9001` was configured to support SSL connections.

   - For remote mode connection to a Gateway daemon, follow these steps:
     - Set `connectionURL` to specify `ssl` as the protocol and the host name of the Gateway daemon, for example, `ssl://lp01.redbooks.ibm.com`.
     - Set `portNumber` to specify the Gateway daemon SSL protocol handler port, for example, `8070`.
   - Optional: Restrict the cipher suites to be used on the Cipher Suites property.
   - Leave other properties unchanged.

For more information about how to configure SSL between WebSphere Application Server on Multiplatforms using local mode IPIC connection to CICS TS and SSL between a Java client and a Gateway daemon, see the CICS TG Information Center:

http://pic.dhe.ibm.com/infocenter/cicstgmp/v9r0/index.jsp

12.11 Transaction tracking with Cross Component Trace

You can use Cross Component Trace (XCT) to track individual requests as they flow between WebSphere Application Server, CICS TG, and CICS TS, assisting you with both problem diagnosis and system planning and configuration. The XCT facility is available when using IPIC connections and WebSphere Application Server Version 8.5 or later. As a request flows through the system, the related XCT information is recorded in the WebSphere High Performance Extensible Logging (HPEL) log, CICS TG request monitoring exit data, and CICS task association data.

To enable XCT in WebSphere Application Server, use the following steps:

1. Log in to the WebSphere Application Server administration console.
2. Click the application server name, for example, `itsoServer01`, from the **Troubleshooting** → **Logs and trace** section of the navigation menu.
3. Enable High Performance Extensible Logging (HPEL):
   a. Click **Switch to HPEL Mode**. This option will not be displayed if HPEL is already enabled.
   b. Save the changes to the master configuration.
4. Enable XCT:
   a. After saving the information in the Switch to HPEL Mode dialog box, the Change log detail levels dialog box for the application server is displayed. Alternatively, you can open this dialog box by navigating to **Troubleshooting** → **Logs and trace** → ***serverName*** → **Change log detail levels**.
   b. In the Correlation section, select **Enable log and trace correlation** and **Include request IDs in log and trace records and create correlation log records**.
   c. Save the changes to the master configuration.
5. If you are using a clustered WebSphere Application Server configuration, repeat these steps for all application servers in the cluster, and synchronize the changes to the nodes.

The XCT information in WebSphere Application Server is written to the HPEL log. The XCT information can be viewed in several ways:

- To view XCT information using the administration console, use the following options:
  a. Click **Troubleshooting** → **Logs and trace** → ***serverName*** → **View HPEL logs and trace** from the navigation menu.
  b. Optional: Expand **Content and Filtering Details** to filter the HPEL log records displayed. For example, to view only entries for the ECI resource adapter, set Include logger to `com.ibm.websphere.XCT` and set Message contents to `ECIRA*`.
- To view XCT information using the command line, use the following options:

  Use the WebSphere Application Server **`logViewer.bat`** command or **`logViewer.sh`** script. For information about using the command, use the **`-help`** option on the command line, or see the WebSphere Application Server Information Center:

  http://ibm.co/1qPvJzP

  For example, to see only entries for the ECI resource adapter, use the following syntax:

  WAS_HOME*/profiles/*profileName*/bin/logviewer.sh -format advanced -message *ECIRA
- Alternatively, use the IBM WebSphere Cross Component Trace Logviewer, which is available as a plug-in to the IBM Support Assistant:
  a. Download the IBM Support Assistant (ISA) Workbench from this website:

     http://www.ibm.com/software/support/isa
  b. Start the ISA Workbench, and then click **Update** → **Find New** → **Tools Add-ons**.
  c. Select the tool **JVM-based Tools** → **IBM WebSphere Cross Component Trace Logviewer**. Follow the prompts to complete the installation.
  d. Restart the ISA Workbench. Then, follow the instructions that are provided with the Logviewer to view the XCT entries in the HPEL logs.

For more information about using XCT with CICS TG and example output from WebSphere Application Server, CICS TG request monitoring, and CICS TS task association data, see the CICS TG Information Center:

http://ibm.co/1mpkdan

13

High availability configuration

This chapter describes the steps to implement a high availability (HA) system for a 3-tier remote mode scenario, where a Java EE application in WebSphere Application Server uses the ECI resource adapter to connect to z/OS Gateway daemons and the Gateway daemons connect to CICS Transaction Server (CICS TS) regions using the Internet Protocol interconnectivity (IPIC) connection protocol.

The configuration described in this chapter uses options that are specific to the CICS Transaction Gateway for z/OS product (CICS TG). For a description of the high availability concepts, which includes options available for CICS TG for Multiplatforms, see Chapter 7, “High availability concepts” on page 103.

We use Sysplex Distributor and TCP/IP port sharing to distribute requests from WebSphere Application Server to multiple z/OS Gateway daemons configured as a Gateway group across two logical partitions (LPARs) of a z/OS Parallel Sysplex. Dynamic server selection (DSS) is used to distribute the workload to different CICS terminal-owning regions (TORs).

This chapter contains the following topics:

13.1 High availability use case

The use case for this scenario is to provide scalability and failover for the 3-tier remote mode topology, where a Java EE application in WebSphere Application Server on Multiplatforms uses the ECI resource adapter to connect to a highly available Gateway group. IPIC connections are used between the Gateway daemons and the CICS TS regions.

The Java EE application workload is driven by a total of 200 users, with an expected number of 100 concurrent requests.

In normal operating conditions, the workload needs to be equally distributed across the system components. However, the configuration needs to be able to cope with the following failure scenarios: loss of a single CICS region, loss of a single Gateway daemon, or loss of a z/OS LPAR. In the worst case scenario, each Gateway daemon must be capable of handling the entire workload.

The CICS TG JCA resource adapter installation verification test (`IVT`) will be used. The `IVT` is a Java EE application that provides a web interface and makes both non-transactional and transactional ECI calls to CICS using a Java EE Connector Architecture connection factory associated with the ECI resource adapter. The workload is driven using Rational Performance Tester, details of which are beyond the scope of this chapter.

13.2 High availability topology

This section describes the high availability topology used in this chapter.

The high availability environment used in this chapter, as shown in Figure 13-1 on page 297, includes the following key components:

- A z/OS Parallel Sysplex with two LPARs. Each LPAR hosts multiple Gateway daemons and CICS TS regions.
- Each Gateway daemon is configured to connect to all TORs using IPIC.
- Dynamic server selection provides a flexible mechanism for dynamically controlling the flow of work to the CICS regions.
- A highly available Gateway group provides workload balancing and failover between the Gateway daemons.
- Sysplex Distributor and TCP/IP port sharing are used to distribute ECI requests from WebSphere Application Server to the Gateway group.
- A WebSphere Application Server cluster on AIX hosts the Java EE application, which flows the ECI requests.

Each of these components will be discussed in more detail in the following sections.

This chapter does not discuss Secure Sockets Layer (SSL), but the SSL configurations described in previous chapters are applicable to this high availability scenario.

Figure 13-1 on page 297 shows the high availability environment implemented in this chapter.

Note: All the Gateway daemons are connected to all the CICS TS regions, but not all of these connections are indicated in the diagram, to simplify the image.

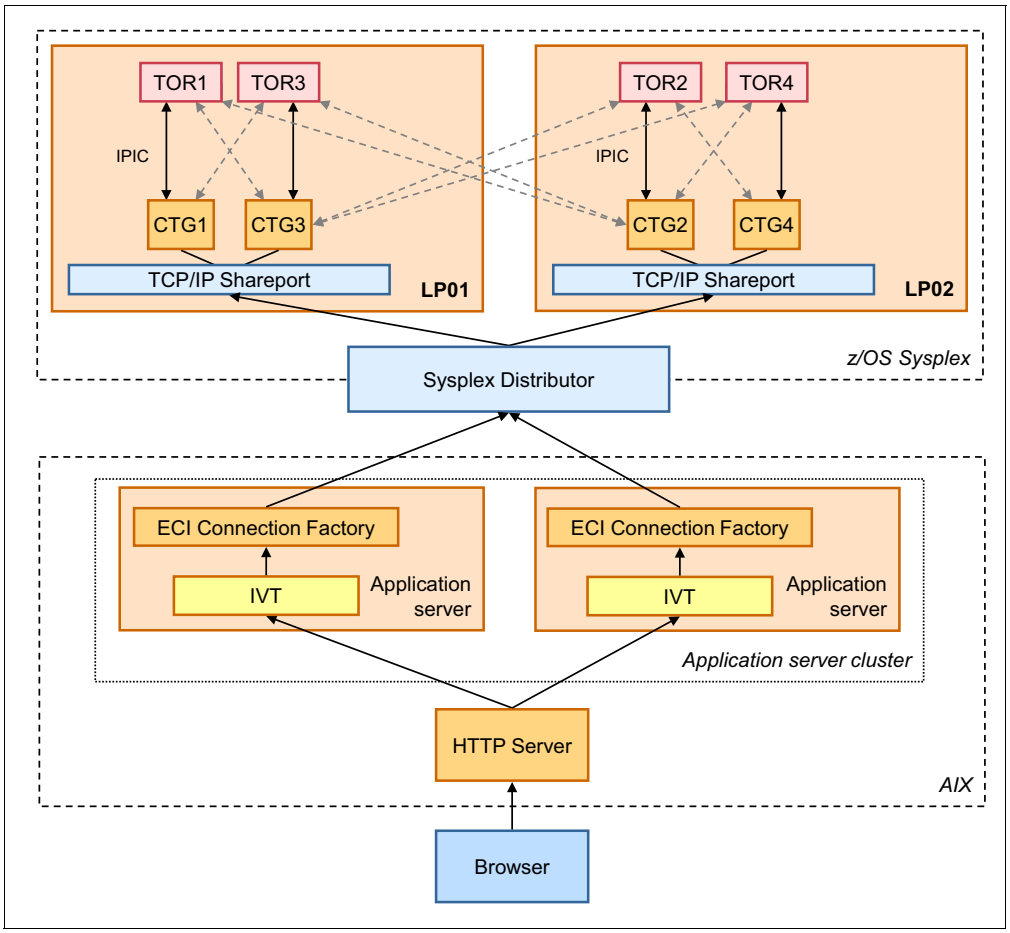


Figure 13-1 High availability topology implemented in this chapter

Assumptions

The instructions to implement the high availability environment described in Figure 13-1 make the following assumptions:

- Four CICS TS regions shown as `TOR1`, `TOR2`, `TOR3`, and `TOR4` are active across the two LPARs (`LP01` and `LP02`) of the Parallel Sysplex. Each CICS TS region has the sample server programs `EC01` and `EC03` installed. Each CICS TS region is configured with predefined IPIC connections:
  - The CICS TS region configuration is based on Chapter 9, “Configuring IPIC in CICS Transaction Gateway for z/OS” on page 147, scaled to provide four TORs.
  - The `TCPIPSERVICE`s are configured to listen on the static virtual IP addresses (VIPAs) `lp01.redbooks.ibm.com` (`192.168.0.1`) and `lp02.redbooks.ibm.com` (`192.168.0.2`).
  - The `IPCONN` resource definition `receivecount` is increased to `100`, so that in the worse case scenario, a single TOR is capable of handling the entire anticipated workload.
- A WebSphere Application Server cluster was configured on AIX. The ECI resource adapter was installed, the Java EE Connector Architecture connection factory was configured, and the `IVT` application was installed. See Chapter 12, “Integrating with WebSphere Application Server” on page 261.
- 4 Gateway daemons, shown as `CTG1`, `CTG2`, `CTG3`, and `CTG4`, all listen for new connections on a common port, `2006`, and are bound to distributed dynamic VIPA (DVIPA), `lp00.redbooks.ibm.com` (`1.2.3.4`).

Table 13-1 lists the resources and values used to configure the high availability topology described in this chapter.

Table 13-1 HA topology resource summary

Component	Attribute	Value
z/OS TCP/IP	LP01 Hostname/IP address	`lp01.redbooks.ibm.com/192.168.0.1`
z/OS TCP/IP	LP02 Hostname/IP address	`lp02.redbooks.ibm.com/192.168.0.2`
Sysplex Distributor	DVIPA Hostname/IP address	`lp00.redbooks.ibm.com/1.2.3.4`
Sysplex Distributor	Ports in port list	`2006`
TCP/IP Profile	Shared port definition port	`2006`
TCP/IP Profile	Shared port definition jobname	`CTG*`
TCP/IP Profile	Dynamic cross-system coupling facility (XCF) address (LPAR)	`10.1.100.1 (LP01)` `10.1.100.2 (LP02)`

Gateway daemon	Jobnames/APPLIDs (owning LPAR)	`CTG1ITSO/ITSOCTG1 (LP01)`
		`CTG2ITSO/ITSOCTG2 (LP02)`
		`CTG3ITSO/ITSOCTG3 (LP01)`
		`CTG4ITSO/ITSOCTG4 (LP02)`

Gateway daemon	APPLIDQUALIFIER *all Gateway daemons*	`CTGALP00`
Gateway daemon	`tcp` protocol handler bind address/port *all Gateway daemons*	`lp00.redbooks.ibm.com/2006`
Gateway daemon	DSSPOLICY Mapping name	`CICSTOR`

Gateway daemon	IPICSERVER name/CICS TCPIPSERVICE address/port
	all Gateway daemons
		`CICSTOR1/lp01.redbooks.ibm.com/9011`
		`CICSTOR2/lp02.redbooks.ibm.com/9012`
		`CICSTOR3/lp01.redbooks.ibm.com/9013`
		`CICSTOR4/lp02.redbooks.ibm.com/9014`

Gateway daemon	`statsapi` protocol handler bind address/port (Gateway daemon)	`lp01.redbooks.ibm.com/2981 (CTG1ITSO)`
		`lp02.redbooks.ibm.com/2982 (CTG2ITSO)`
		`lp01.redbooks.ibm.com/2983 (CTG3ITSO)`
		`lp02.redbooks.ibm.com/2984 (CTG4ITSO)`

CICS TS	CICS region APPLID values (owning LPAR)	`S20351T1 (LP01)`
		`S20351T2 (LP02)`
		`S20351T3 (LP01)`
		`S20351T4 (LP02)`

CICS TS	TCPIPSERVICE definition hostname/port (owning CICS region)	`lp01.redbooks.ibm.com/9011 (S2035T1)`
		`lp02.redbooks.ibm.com/9012 (S2035T2)`
		`lp01.redbooks.ibm.com/9013 (S2035T3)`
		`lp02.redbooks.ibm.com/9014 (S2035T4)`

CICS TS	IPCONN definition ReceiveCount	`100`

13.3 Connection balancing

This section describes how balancing TCP/IP requests into the z/OS Gateway daemons is achieved in the high availability environment described in this chapter.

Connection balancing of TCP/IP requests from the client application is achieved by using TCP/IP port sharing to distribute requests across the Gateway daemons in the same LPAR and using Sysplex Distributor to distribute requests across the Parallel Sysplex. For more information about the concepts of connection balancing, see 7.2, “Connection balancing” on page 105.

The Gateway daemon health reporting information is used in calculating the IBM z/OS Workload Manager (WLM) health. WLM health can be used as an additional mechanism to influence the TCP/IP load balancing algorithm, in both TCP/IP port sharing and Sysplex Distributor. This causes requests to be proportionally distributed to the healthier Gateway daemons.

Perform these tasks to use the z/OS Gateway daemon and WLM health information for load balancing decisions:

- For Sysplex Distributor, set `SERVERWLM` on the `VIPADISTRIBUTE` statement.
- For TCP/IP port sharing, set `SHAREPORTWLM` on the TCP/IP profile `PORT` definition.

For more details about how WLM server-specific recommendations are calculated and used to influence the load balancing algorithm, see 7.2.2, “z/OS shared ports” on page 107.

The information about how this is implemented in this scenario is described in 13.3.1, “Sysplex Distributor” on page 300 and 13.3.2, “Enabling WLM recommendations for TCP/IP port sharing” on page 304.

Figure 13-2 on page 300 shows the Sysplex Distributor and TCP/IP port sharing environment configuration used in this chapter.

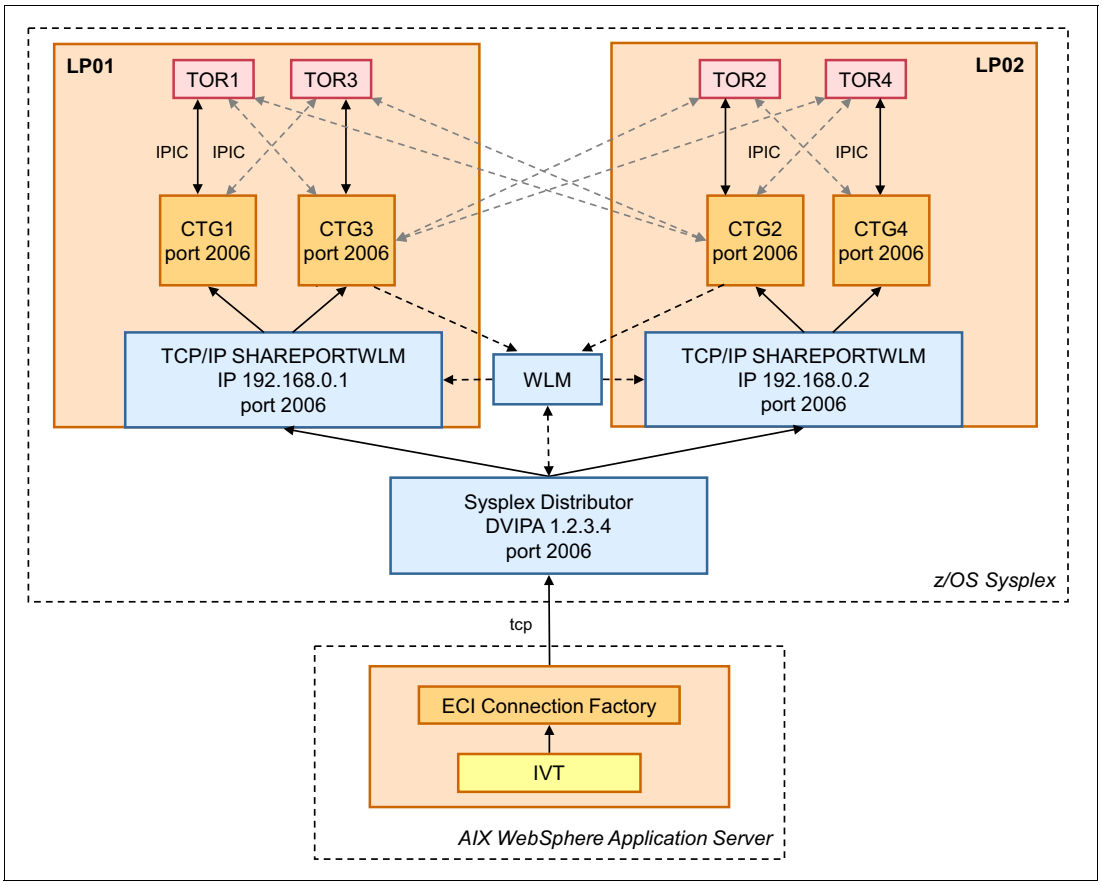


Figure 13-2 Sysplex Distributor and TCP/IP port sharing configuration

Binding of listener ports

In this implementation, we want to ensure that only those client applications using the advertised DVIPA address can connect to a Gateway daemon. So, the `tcp` protocol handler definition specifies a bind address of `lp00.redbooks.ibm.com` (`1.2.3.4`). In this way, external application programs do not need any knowledge of the LPAR static VIPA, but even if they have this knowledge, they cannot bypass the Sysplex Distributor by attaching directly to the Gateway daemon.

By contrast, the CICS `TCPIPService` definitions bind to the static VIPA for their LPAR. The CICS `TCPIPService` listener ports are not shared (they must not be for CICS TG IPIC connections), and are of no interest to the external applications. Only the `IPICServer` definitions in the Gateway daemon configuration file need to reference the address and port numbers associated with the `TCPIPService` definitions. Similarly, the statistics API protocol handlers for each Gateway daemon must not be shared, and so, they are also bound to the static VIPA for their LPAR.

13.3.1 Sysplex Distributor

In a high availability environment with multiple Gateway daemons configured across different LPARs in a Parallel Sysplex, Sysplex Distributor can provide a single system image view of the TCP/IP stacks in the individual LPARs. Sysplex Distributor defines an IP address (dynamic VIPA) for this single system image, which client applications can use to connect to the Gateway daemons.

When client applications attempt to establish a connection to a z/OS Gateway daemon's advertised IP address and port, Sysplex Distributor distributes requests to one of the available Gateway daemons within the single system image. After a connection is established, requests from the client application continue to use the same connection.

> **Note:** One important aspect of using Sysplex Distributor is that the instances of the Gateway daemon must be functional clones, for example, configured as a highly available Gateway group. Also, the Gateway daemons must be able to handle any XA transaction affinities that can arise during the execution of a client request.

In the high availability environment shown in Figure 13-1 on page 297, TCP/IP port sharing is used to distribute requests across multiple Gateway daemons within the same LPAR. Sysplex Distributor is used to distribute the client requests across the TCP/IP stacks of the individual LPARs. Sysplex Distributor will be configured to use the z/OS Gateway daemon and WLM health information for its load balancing decisions. The Sysplex Distributor dynamic VIPA hostname is `lp00.redbooks.ibm.com`, and the IP address is `1.2.3.4`.

Configuring Sysplex Distributor

Follow these steps to use Sysplex Distributor to distribute requests between multiple Gateway daemons in different LPARs of a Parallel Sysplex:

1. Install and configure Sysplex Distributor and its prerequisites on all LPARs in the Parallel Sysplex that hosts the Gateway daemons.

   For details about installing and configuring Sysplex Distributor, see the z/OS Communications Server Information Center:

   http://ibm.co/1uOWrLk

2. Enable dynamic Cross-System Coupling Facility (XCF), which is a requirement of Sysplex Distributor, on each host. The dynamic XCF IP addresses are used by the distributing TCP/IP host when it redirects packets to a target host.

   > **Note:** Dynamic XCF addresses are used internally, and they are not to be referenced directly by the CICS TG or CICS TS configuration. Do not advertise dynamic XCF addresses to external applications.

   `IPCONFIG` statements in the TCP/IP profile data are used to defined dynamic XCF IP addresses. Example 13-1 shows the configuration statements to enable dynamic XCF address `10.100.0.1` on LPAR `LP01`.

   Example 13-1 Enable dynamic XCF on LPAR LP01

   ```
   IPCONFIG
     DATAGRamfwd
     DYNAMICXCF 10.100.0.1 255.255.255.0 1
     SYSPLEXRouting
     IGNORERedirect
     SOURCEVIPA
   ```

   Example 13-2 on page 302 shows the configuration statements to enable dynamic XCF for static VIPA address `10.100.0.2` on LPAR `LP02`.

Example 13-2 Enable dynamic XCF on LPAR LP02

```
IPCONFIG
  DATAGRamfwd
  DYNAMICXCF 10.100.0.2 255.255.255.0 1
  SYSPLEXRouting
  IGNORERedirect
   VARSUBNETTING
```

3. Configure Sysplex Distributor to use the z/OS Gateway daemon and WLM health information for load balancing decisions.

   Add configuration statements in the TCP/IP profile data set as shown in the following examples.

   Example 13-3 shows the configuration statements on LPAR `LP01` to define the advertised DVIPA address `1.2.3.4` to distribute requests received on port `2006` across the LPARs with dynamic XCF addresses `10.100.0.1` (`LP01`) and `10.100.0.2` (`LP02`), using the server-specific WLM recommendations (keyword `SERVERWLM`).

   Example 13-3 Define DVIPA for port 2006 on LPAR LP01 using SERVERWLM

```
VIPADYNAMIC
VIPADEFINE MOVEABLE IMMEDIATE 255.255.252.0  1.2.3.4
VIPADISTRIBUTE DEFINE DISTMETHOD SERVERWLM 1.2.3.4
   PORT 2006
   DESTIP 10.100.0.1 10.100.0.2
ENDVIPADYNAMIC
```

   Example 13-4 shows the configuration statements on LPAR `LP02` to define the advertised DVIPA address `1.2.3.4` to distribute requests received on port `2006` across the LPARs with dynamic XCF addresses `10.100.0.2` (`LP02`) and `10.100.0.1` (`LP01`), using the server-specific WLM recommendations (keyword `SERVERWLM`).

   Example 13-4 Define DVIPA for port 2006 on LPAR LP02 using SERVERWLM

```
VIPADYNAMIC
VIPADEFINE MOVEABLE IMMEDIATE 255.255.252.0  1.2.3.4
VIPADISTRIBUTE DEFINE DISTMETHOD SERVERWLM 1.2.3.4
   PORT 2006
   DESTIP 10.100.0.2 10.100.0.1
ENDVIPADYNAMIC
```

In this chapter, port `2006` on the DVIPA is explicitly defined to distribute connections between `LP01` and `LP02`. This allows the CICS TG remote application to connect with one of the four individual Gateway daemon `TCP` protocol handler ports. In addition, if the Gateway daemons define an `SSL` protocol handler, a second `PORT` statement (for example, `PORT 8050`) is added to the `VIPDISTRIBUTE` definition.

Note: An alternative approach to explicitly listing individual port numbers is to omit the **`PORT`** parameter from the Sysplex Distributor `VIPADISTRIBUTE` statement. In this case, any server binding a listening port to the DVIPA is eligible for connection workload balancing.

In Example 13-3 on page 302 and Example 13-4 on page 302, the DESTIP statement on the VIPADISTRIBUTE definition specifies the dynamic XCF addresses (as defined on the IPCONFIG DYNAMICXCF statement) of the TCP/IP stacks in the sysplex that are to be target stacks for the dynamic VIPA. Alternatively, DESTIP ALL can be specified, causing all TCP/IP stacks in the sysplex that have defined a dynamic XCF address of the same type as the IP address specified in this VIPADISTRIBUTE statement to be defined as target stacks for the dynamic VIPA.

If only one server binds to the distributed DVIPA and port and establishes a listening socket, that server will get all of the work. When a second server binds to the distributed DVIPA using the same port and establishes a listening socket, it will immediately become eligible to participate in connection workload balancing.

For more information about using and configuring Sysplex Distributor, see *IBM z/OS V1R13 Communications Server TCP/IP Implementation: Volume 3 High Availability, Scalability, and Performance*, SG24-7998, which is available at this website:

http://www.redbooks.ibm.com/abstracts/sg247998.html

Verifying the Sysplex Distributor definitions

In this book, Sysplex Distributor is configured to be active on one LPAR with the other LPAR defined as a backup, so that if the distributing stack fails, the backup can take over the distributing function.

For this chapter, the advertised DVIPA address is active on LPAR LP01 with the other LPAR LP02 able to act as a backup. This can be shown by issuing the following command from the z/OS System Display and Search Facility (SDSF) panel on each LPAR:

```
d tcpip,tcpip,netstat,vipadyn
```

Example 13-5 shows the output on LPAR LP01 showing that DVIPA is active.

Example 13-5 LPAR LP01 is configured as the active stack for Sysplex Distributor

```
DYNAMIC VIPA:
  IP ADDRESS   ADDRESSMASK      STATUS   ORIGINATION    DISTSTAT
  1.2.3.4      255.255.252.0    ACTIVE   VIPADEFINE     DIST/DEST
```

Example 13-6 shows the output on LPAR LP02 showing that DVIPA is a backup.

Example 13-6 LPAR LP02 is defined as a backup stack for Sysplex Distributor

```
DYNAMIC VIPA:
  IP ADDRESS   ADDRESSMASK      STATUS   ORIGINATION    DISTSTAT
  1.2.3.4      255.255.252.0    BACKUP   VIPADEFINE     DIST/DEST
```

To view where requests to port 2006 are being routed, issue the following command from the z/OS SDSF panel on the active LPAR (LP01):

```
d tcpip,tcpip,netstat,vdpt
```

Example 13-7 on page 304 shows the output from this command. The equal WLM values indicate an equal weighting between the LPARs. The TOTALCONN values show that an equal number of requests were distributed to each LPAR.

Example 13-7 Sysplex Distributor request distribution

```
DYNAMIC VIPA DESTINATION PORT TABLE FOR TCP/IP STACKS:
DEST IPADDR     DPORT  DESTXCF ADDR    RDY    TOTALCONN      WLM TSR FLG
1.2.3.4        02006   192.168.0.1      002   0000000015     01    100
1.2.3.4        02006   192.168.0.2      002   0000000015     01    100
```

For a high-level summary of the configuration, use the TSO **netstat** command, suboption **home**:

```
d tcpip,tcpip,netstat,home
```

Example 13-8 A summary of the configuration from the netstat home command

```
HOME ADDRESS LIST:
ADDRESS          LINK             FLG
192.168.0.1      OSA2040LNK       P
10.1.100.1       EZASAMEMVS
10.1.100.1       EZAXCF02
1.2.3.4          VIPL090C0444
127.0.0.1        LOOPBACK
5 OF 5 RECORDS DISPLAYED
```

The second column in the output from the **netstat home** command, shown in Example 13-8, provides each address entry with an indication of its type. A link value beginning with the characters "OSA" indicates that the address is associated with a physical network adapter. The entries beginning with "EZA" come from dynamic XCF addresses. The entry beginning with "VIPL" is for the externally advertised DVIPA.

13.3.2 Enabling WLM recommendations for TCP/IP port sharing

TCP/IP port sharing is used in high availability environments where multiple Gateway daemons are configured in the same LPAR to provide a single IP address and port for client applications to use to connect to the Gateway daemons.

When client applications attempt to establish a connection to a z/OS Gateway daemon's advertised IP address and port, TCP/IP port sharing distributes connections to one of the available Gateway daemons listening on the shared port in that LPAR. After a connection is established, requests from the client application continue to use the same connection.

> **Note:** One important aspect of using TCP/IP port sharing is that the instances of the Gateway daemon must be functional clones, for example, configured as a highly available Gateway group. Also, the Gateway daemons must be able to handle any XA transaction affinities that can arise during the execution of a client request.

In the high availability environment shown in Figure 13-1 on page 297, TCP/IP port sharing uses a WLM server-specific recommendation. This includes the z/OS Gateway daemon health reporting information in the workload balancing algorithm.

To enable TCP/IP port sharing to use the Gateway daemon health information for load balancing decisions, set SHAREPORTWLM on the TCP/IP profile PORT definition, as shown in Example 13-9 on page 305.

Example 13-9 TCP/IP profile SHAREPORTWLM definition

```
PORT
      2006 TCP CTG* SHAREPORTWLM
```

The steps to enable port sharing based on WLM server-specific recommendations are described in “Configuring WLM recommendations for TCP/IP port sharing” on page 305.

CICS TG health reporting

The z/OS Gateway daemons can monitor certain error codes to determine the health of communications with CICS TS. When CICS TG health reporting is enabled, the Gateway daemon health is initially set to `100` and it is calculated to represent the percentage of requests that succeeded during the health interval. Therefore, ECI requests through a Gateway daemon that fail to complete successfully because the CICS TS server is unavailable cause the health of the Gateway daemon to be reduced. The minimum value is `0`. For more details, see “CICS TG WLM health” on page 110.

In the high availability environment shown in Figure 13-1 on page 297, health reporting is enabled in the Gateway daemons by setting the following configuration file parameters:

- `healthreporting=on`
- `healthinterval=60` (the default value)

When the Gateway daemon is started with health reporting enabled, the following message is written to the `STDOUT` log:

```
CTG6422I Health reporting is enabled.
```

Configuring WLM recommendations for TCP/IP port sharing

Use the following steps to set up TCP/IP port sharing based on WLM server-specific recommendations to distribute requests to the Gateway daemon `tcp` protocol handler port `2006`, between multiple Gateway daemons in the same LPAR:

1. Check that port `2006` is not currently reserved.

   Display ports that are currently reserved:

   – Either issue the SDSF command: **`/D TCPIP,,N,PORTL`**
   – Or, issue the UNIX System Services command: **`netstat -o`**

   If the port is currently registered as a shared port but not based on WLM server-specific recommendations, the Flags attribute is `DAS` instead of `DASW` as shown in Example 13-10. In this case, delete the existing TCP/IP profile definition, as described in 13.3.4, “Modifying or deleting a TCPIP port sharing definition” on page 308.

   Example 13-10 Example output for TCP/IP shared port not using WLM

```
PORT#    PROT    USER       FLAGS    RANGE       SAF NAME
2006     TCP     CTG*       DAS
```

2. Add a `PORT` entry for port `2006` to the TCP/IP profile for the LPAR as a shared port using WLM server-specific recommendations.

   By adding the entry to the TCP/IP profile, the definition will be automatically activated after a TCP/IP restart or IPL of the LPAR.

   – The TCP/IP profile will probably already contain the `PORT` statement followed by definitions for existing reserved ports. Add the statement for the Gateway daemon port `2006` as shown in Example 13-11. This example adds a definition for port `2006` to be reserved as a TCP/IP shared port using WLM server-specific recommendations for z/OS jobnames starting with `CTG` (notice the asterisk (*) at the end of the name), that is, the jobnames used by the Gateway daemon address spaces.

   Example 13-11 Example definition to add a TCP/IP shared port using WLM

```
PORT
   2006 TCP CTG* SHAREPORTWLM ; Gateway daemons
```

3. Repeat these steps on each LPAR where TCP/IP port sharing is to be activated for Gateway daemons in the Gateway group.

13.3.3 Testing TCP/IP port sharing using WLM

Use the following steps to test that TCP/IP port sharing using WLM works correctly:

1. Display the reserved ports:
   – Either issue the SDSF command: **`/D TCPIP,,N,PORTL`**
   – Or, issue the UNIX System Services command: **`netstat -o`**

   The output is in the format shown in Example 13-12, where `SW` in the Flags value indicates that the port is shared and that the port uses WLM server-specific recommendations.

   Example 13-12 Example netstat output for a TCP/IP shared port using WLM

```
PORT#    PROT     USER       FLAGS    RANGE       SAF NAME
2006     TCP      CTG*       DASW
```

2. Start the Gateway daemons:
   – Ensure that the Gateway daemon jobnames start with the value defined in the shared port definition, for example, `CTG`. They need to start successfully and listen on the shared port.

Note: If you start a Gateway daemon listening on a reserved port with a jobname that does not match the value defined in the TCP/IP profile for the shared port, the Gateway daemon fails to start with the following message written to the `STDERR` log:

```
"CTG6525E Unable to start handler for the tcp: protocol, port: 2006, because:
java.net.BindException: EDC5111I Permission denied"
```

3. Display the WLM information:
   - Either issue the TSO command: **NETSTAT ALL (PORT 2006**
   - Or, issue the UNIX System Services command: **netstat -A -P *2006***

     The output contains an entry for each Gateway daemon using the WLM shared port, where the "Client Name" is the Gateway daemon jobname. The WLM information is displayed at the bottom of each output entry. Example 13-13 displays example output information, immediately after starting a Gateway daemon to use a TCP/IP shared port.

     Note: This is an extract of the WLM information from all the information displayed by the **netstat** command.

     Example 13-13 Example netstat output showing initial WLM health information

```
SharePort:          WLM
            RawWeight:        000 NormalizedWeight:   000
            Abnorm:          0000 Health:             000
```

4. Send an ECI request in remote mode to the shared port (for example, using the CICS TG `EciB1` sample). Repeat for the number of Gateway daemons listening on the shared port. The requests round-robin to each Gateway daemon because they all start with an equal health weighting.
5. Display the Health information for CICS TG and WLM:
   - Issue the SDSF command: **F *jobname*,APPL=STATS,GS=GD_CHEALTH**
   - Issue the TSO command: **NETSTAT ALL (PORT 2006** or issue the UNIX System Services command: **netstat -A -P *2006***

     Assuming that the target CICS server was available and the requests succeeded, the health reported by the CICS TG statistic and the WLM `Health` value for each Gateway daemon are both `100`. The WLM `NormalizedWeight` starting values for each Gateway daemon start with the same value (that is, have an equal weighting) as shown in Example 13-14.

     Example 13-14 Example netstat output showing WLM health values of 100

```
Output for Gateway daemon 1 GD_CHEALTH=100:
SharePort:       WLM
        RawWeight: 000    NormalizedWeight:006
        Abnorm:  0000     Health:          100

Output for Gateway daemon 2 GD_CHEALTH=100:
SharePort:       WLM
        RawWeight: 000    NormalizedWeight:006
        Abnorm:  0000     Health:          100
```

Example 13-15 on page 308 shows sample output from the UNIX System Services command, **netstat -A -P *2006***, in a scenario with two Gateway daemons, where all requests in the first Gateway daemon are successful, but 20% of the requests in the second Gateway daemon failed during the last health interval, resulting in a `Health` value of 80%. Also, the `NormalizedWeight` values have been adjusted to advise a 5:4 ratio for distributing requests in favor of the first Gateway daemon.

Example 13-15 Sample netstat output showing WLM health balancing

```
Output for Gateway daemon 1 GD_CHEALTH=100:
SharePort:       WLM
        RawWeight: 000    NormalizedWeight:005
        Abnorm:  0000     Health:          100

Output for Gateway daemon 2 GD_CHEALTH=80:
SharePort:       WLM
        RawWeight: 000    NormalizedWeight:004
        Abnorm:  0000     Health:          080
```

13.3.4 Modifying or deleting a TCPIP port sharing definition

If you no longer want to use TCP/IP port sharing or you want to change an existing shared port definition, for example, to switch between `SHAREPORT` and `SHAREPORTWLM`, delete the existing shared port definition. It is possible to dynamically modify or delete a port sharing definition using the TCPIP **OBEY** command. The **OBEY** command reads the details of the required change from a partitioned data set (PDS) member.

Use the following steps to delete the shared port definition for port `2006` associated with jobnames that start with `CTG*`:

1. Shut down any Gateway daemons using the shared port.
2. Add a `DELETE PORT` entry for shared port `2006` in a PDS member:
   - Create a new member in a z/OS data set, with the attributes `RECFM=FB` and `LRECL=80`, for example:

     `REDBOOKS.TCPIP.PARMS(DPORTWLM)`
   - Ensure that the port, jobname, and `SHAREPORTWLM` or `SHAREPORT` values match those values in the equivalent add port definition.

     Example 13-16 shows an example definition to delete a TCP/IP shared port currently defined using `SHAREPORTWLM` (WLM server-specific recommendations).

     Example 13-16 Example definition to delete a TCP/IP shared port using WLM

     ```
     ; Delete Shared Port 2006 using WLM used by
     ; jobs with JOBNAME CTG*
     DELETE PORT 2006 TCP CTG* SHAREPORTWLM
     ```

     Example 13-17 shows an example definition to delete a TCP/IP shared port currently defined using `SHAREPORT`.

     Example 13-17 Example definition to delete a TCP/IP shared port

     ```
     ; Delete Shared Port 2006 used by
     ; jobs with JOBNAME CTG*
     DELETE PORT 2006 TCP CTG* SHAREPORT
     ```
3. Delete the shared port definition:
   - Issue the SDSF command:

     /V TCPIP,,OBEY,DSN= REDBOOKS.TCPIP.PARMS(DPORTWLM)

     The response displays:

     `EZZ0053I COMMAND VARY OBEY COMPLETED SUCCESSFULLY`

4. Check that the shared port definition is no longer active by using one of the following commands:
   - From SDSF, issue the display command: **/D TCPIP,,N,PORTL**
   - From UNIX System Services (OMVS), issue the **netstat** command: **netstat -o**

   The output from either command will not include an entry for port 2006.

13.4 Highly available Gateway group

A highly available (HA) Gateway group uses TCP/IP load balancing. An HA Gateway group can be regarded as a single logical Gateway daemon. Gateway daemons are defined as belonging to a HA Gateway group by a common APPLID qualifier, within the scope of Parallel Sysplex. The APPLID qualifier definition is mandatory for XA support, but it is a preferred practice to always configure each Gateway daemon with a fully qualified APPLID. Naming conventions and the use of fully qualified APPLIDs must also be considered when configuring IPIC server connections, uniquely identifying a Gateway daemon instance within an HA Gateway group, and within the scope of the Parallel Sysplex.

Each Gateway daemon in an HA group must have identical configuration details for the following components:

- The tcp and ssl protocol handlers
- IPIC server definitions
- DSSPOLICY definitions
- CICS request exit configuration
- Security settings
- XA support

In the high availability environment shown in Figure 13-1 on page 297, four z/OS Gateway daemons (CTG1, CTG2, CTG3, and CTG4) are configured in a single Gateway group on multiple LPARs in a Parallel Sysplex to provide scalability and failover. The configuration is based on the z/OS Gateway daemon (CTG0) configuration defined in Chapter 10, “CICS Transaction Gateway for z/OS configuration” on page 171. The configuration is extended so that each of the Gateway daemons is configured to connect to all the CICS TS regions (TOR1, TOR2, TOR3, and TOR4) using the IPIC protocol. The Gateway daemons in Figure 13-3 on page 310 summarize the configuration used by the highly available Gateway group.

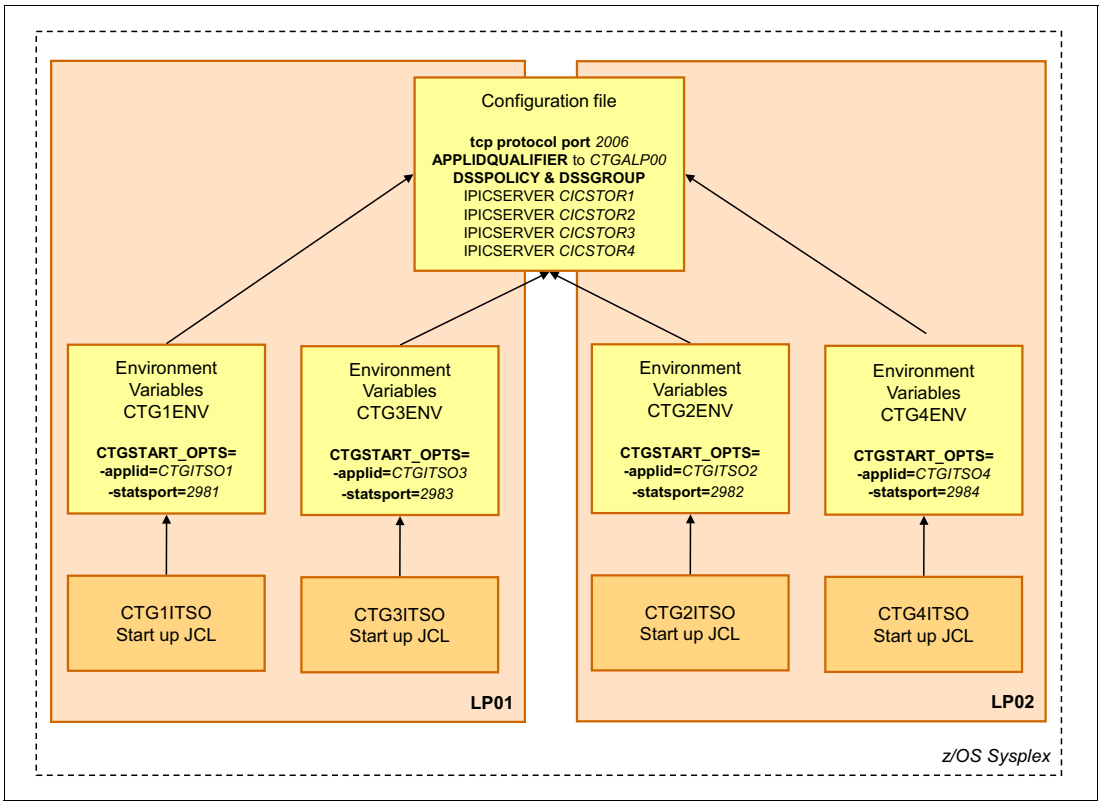


Figure 13-3 Highly available Gateway group configuration

Use the following steps to configure four Gateway daemons in Gateway group `CTGALP00`:

1. Create a single configuration file to be shared between the Gateway daemons in the Gateway group:

   a. Start with the configuration file created in Chapter 10. See 10.1, “Configure the Gateway daemon” on page 172., which defines the `tcp` protocol handler (listening on port `2006`) and the IPIC server definitions. The configuration of DSSPOLICY and CICS request exits are described in 13.5, “Configuring DSSPOLICY” on page 311.

      - Set `InitConnect` to `50`. Expect a potential of 200 user connection requests to be distributed across four Gateway daemons.
      - Set `MaxConnect` to `200`. Allow any of the Gateway daemons to handle a maximum of `200` connection requests in a worse case scenario.
      - Set `InitWorker` to `25`. Expect a potential of 100 concurrent requests to be distributed across four Gateway daemons.
      - Set `MaxWorker` to `100`. Allow any of the Gateway daemons to handle a maximum of 100 concurrent requests in a worse case scenario.

   b. Modify the `tcp` protocol handler **`tcp.parameters`** definition to bind to the DVIPA address `1.2.3.4` defined to Sysplex Distributor in “Configuring Sysplex Distributor” on page 301:

      ```
      protocol@tcp.parameters=bind=lp00.redbooks.ibm.com;port=2006;...
      ```

c. Ensure that the `statsapi` protocol handler in each Gateway daemon is bound to the static VIPA for its LPAR, for example, for `CTG1ITSO` on LPAR `LP01`:

```
protocol@statsapi.parameters=bind=lp01.redbooks.ibm.com;port=2980;...
```

The port number used in this `statsapi` protocol handler definition is not important, because each Gateway will specify an assigned port number, using the **`ctgstart`** option, `statsport`.

> **Note:** If the bind statement for the Statistics API handler is omitted, or a value is omitted (`bind=;`), the special address `inaddrany` is used. In this case, the statistics API handler binds to all the IP addresses, including the DVIPA. The statistics API handler must not bind on a DVIPA address, because there is no sense in distributing these connections.

d. Modify the configuration file to set the Gateway group name via the `APPLIDQUALIFIER` value:

Set `APPLIDQUALIFIER` to `CTGALP00`.

2. Create a unique environment file for each Gateway daemon:
   a. Use the environment file created in Chapter 10. See 10.1, “Configure the Gateway daemon” on page 172, as an example.
   b. Set the **`CICSCLI`** parameter to specify the configuration file.
   c. Set the `CTGSTART_OPTS=` overrides to specify unique values:
      - `-applid=`*CTGITSOn* where *CTGITSOn* is a unique value for each Gateway daemon, for example, `CTGITSO1` or `CTGITSO2`.
      - `-statsport=`*298n* where *298n* is a unique value for each Gateway daemon, for example, `2981` or `2982`.

> **Note:** The statistics API port (`statsport`) must not be shared.

3. Create JCL to start each Gateway daemon as a procedure:
   – Use the JCL created in Chapter 10. See 10.1, “Configure the Gateway daemon” on page 172 as an example, ensuring that it specifies the unique environment file for each Gateway daemon.
   – Set the procedure name as *CTGnITSO*, where *CTGnITSO* is a unique value for each Gateway daemon, for example, `CTG1ITSO` or `CTG2ITSO`.

13.5 Configuring DSSPOLICY

This section describes the configuration of dynamic server selection using DSSPOLICY on z/OS.

13.5.1 Policy based on dynamic server selection (z/OS only)

This section shows the mappings between the logical CICS server name and Gateway groups, applying an algorithm of selection.

The configuration is based on three steps:

1. `GATEWAY` section

   In this section, you must specify the policy, from the list of DSSPOLICY sections defined in the configuration file, that you are going to use. Only one policy can be active at any time. It cannot be changed dynamically at run time, but it can be modified on the subsequent Gateway startup.

   `DSSPOLICY = POLICY1`

   It identifies the active policy.

2. `DSSGROUP` section

   In this section, you can define the CICS servers to be used as a group and the algorithm to be used to route work between them.

   ```
   SECTION DSSGROUP = GROUP1
   Servers = CICSTOR1,CICSTOR2,CICSTOR3,CICSTOR4
   Algorithm = RoundRobin
   ENDSECTION
   ```

   `Servers` identifies the server names of the SECTION IPICSERVER for IPIC of the CICS servers across which we want to distribute the workload.

   > **Note:** In XA transaction support, we cannot mix IPIC and EXCI connected CICS in the `Servers` definition.

   `Algorithm` can be `RoundRobin` or `Failover`. If `RoundRobin` algorithm is selected, the workload is spread, one after the other to the servers identified in the `DSSGROUP`. If `Failover` is selected, only the first CICS server in the list is selected, and the second server will receive the workload only if the first one is unavailable.

   Using the `RoundRobin` algorithm, it is important to consider the parameter **`srvretryinterval`** in the Gateway configuration file. If the CICS server that is currently connected becomes disconnected, an attempt is made to reconnect one second after the CICS server becomes disconnected. If the connection attempt fails, the ECI requests are routed to the second IPIC CICS Server in the list. Any additional attempts on the disconnected CICS are made at the interval specified by the **`srvretryinterval`** parameter. We can have more than one `DSSGROUP` definition with different policies defined.

3. `DSSPOLICY` section

   The `DSSPOLICY` defines the mappings between the logical CICS server name and groups. The client requests (ECI) can contain only a symbolic name to identify the CICS that will receive the call, or it can be absent. In this case, the policy provides the mapping.

   There are a few options that we can use to define the mappings:

   `NONE`: No server name specified by the client application.

   `ANY`: Any server name that does not match an existing mapping. If `ANY` is specified, it is not possible to send a request to a specific `APPLID` anymore.

   `Name`: A specific name.

   ```
   SECTION DSSPOLICY = POLICY1
   SUBSECTION MAPPINGS
   CICSX=GROUP1
   <NONE>=GROUP2
   ```

```
<ANY>=GROUP3
ENDSUBSECTION
ENDSECTION
```

Mappings are applied in order of precedence, from most specific to least specific. In Example 13-18, the mappings in DSSPOLICY PRODPOL1 are applied for ECI requests specifying a CICS server name of CICSTOR, then the <NONE> mapping is applied for ECI requests that omit a CICS server name. Any other ECI request is subject to the <ANY> mapping.

Example 13-18 shows a sample implementation in which we have two policies: one policy for test and one policy for production environment.

There are also four DSSGROUPs with different IPICSERVER definitions and different algorithms.

Example 13-18 Dynamic server selection configuration

```
DSSPOLICY = PRODPOL1  # Active policy

SECTION DSSGROUP = GROUP1
Servers = CICSTOR1,CICSTOR2,CICSTOR3,CICSTOR4
Algorithm = RoundRobin
ENDSECTION
SECTION DSSGROUP = GROUP2
Servers = CICSTOR1,CICSTOR3
Algorithm = RoundRobin
ENDSECTION
SECTION DSSGROUP = GROUP3
Servers = CICSTOR2,CICSTOR4
Algorithm = Failover
ENDSECTION

SECTION DSSPOLICY = PRODPOL1 # Active policy
SUBSECTION MAPPINGS
CICSTOR=GROUP1
<NONE>=GROUP2
<ANY>=GROUP3
ENDSUBSECTION
ENDSECTION
SECTION DSSPOLICY =TESTPOL1 # Backup policy
SUBSECTION MAPPINGS
      CICSX=GROUP4
ENDSUBSECTION
ENDSECTION
```

Figure 13-4 on page 314 shows the ECI requests flow. If the ECI is sent with server name CICSTOR, the request is RoundRobin-routed (1 - 4) to GROUP1 (CICSTOR1, CICSTOR2, CICSTOR3, and CICSTOR4).

If the ECI is sent without a server name, the requested is RoundRobin-routed (5, 6) to GROUP2 (CICSTOR1 and CICSTOR3).

If the ECI is sent with a general server name (not CICSTOR, but not blank), the requested is routed (7) to GROUP3 (CICSTOR2, and if it is not available, to CICSTOR4).

There is also another policy defined, TESTPOL1, but it is not activated. It is only to show that you can have more than one policy, but only one must be active at a time.

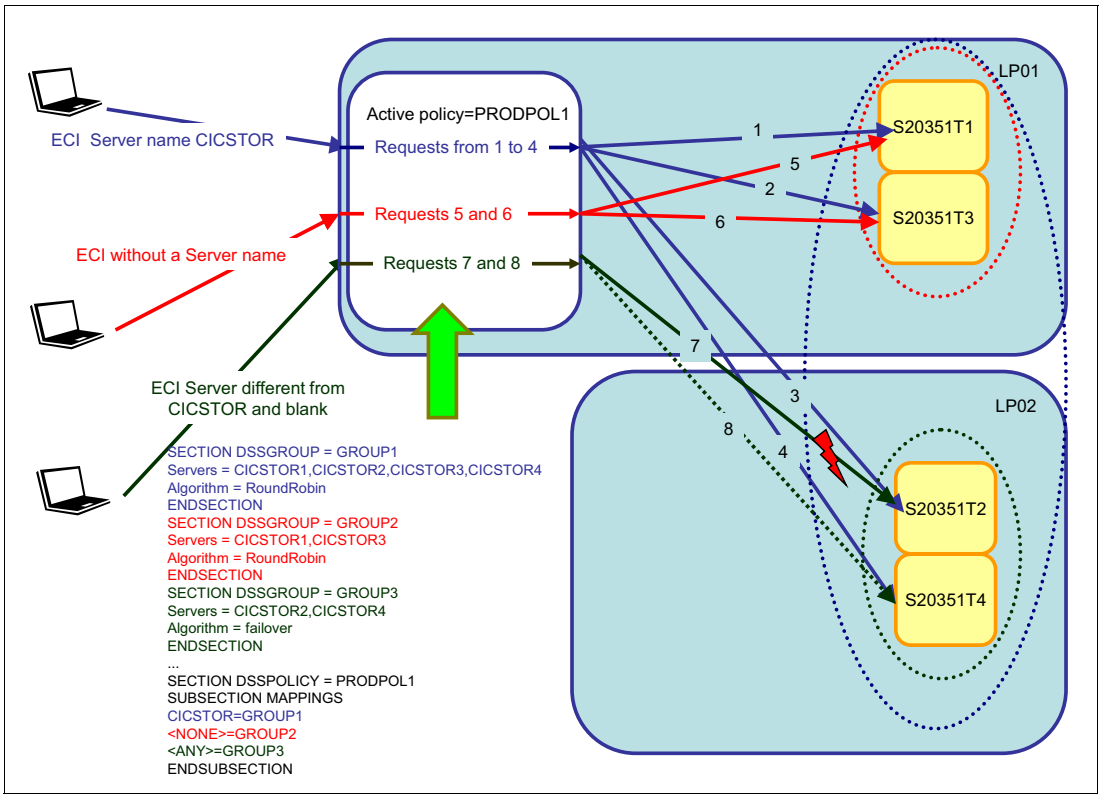


Figure 13-4 Shows the ECI request flows based on the active DSSPOLICY

During the Gateway startup, you will receive the following messages on the console as shown in Example 13-19.

Example 13-19 Message sent to console

```
CTG9903I Current DSS policy is PRODPOL1
   <ANY>=CICSTOR2,CICSTOR4(FAILOVER)
   <NONE>=CICSTOR1,CICSTOR2(ROUNDROBIN)
   CICSTOR=CICSTOR1,CICSTOR2,CICSTOR3,CICSTOR4(ROUNDROBIN)
```

13.6 WebSphere Application Server cluster

This section describes considerations for using WebSphere Application Server clusters with the ECI resource adapter in a high availability environment.

13.6.1 WebSphere Application Server high availability topology

This section focuses on the high availability features provided by CICS TG and considerations for configuring CICS TG in a high availability scenario. Therefore, certain aspects of the WebSphere Application Server configuration are intentionally simplified and only described briefly.

A WebSphere Application Server cluster is a group of cloned application servers that are managed together and participate in workload management. When you install an application into a cluster, the application is automatically installed on each application server in the cluster. Because each cluster member contains the same applications, client requests can be distributed to any of the application servers.

Figure 13-5 shows the WebSphere Application Server cluster configuration used in this chapter.

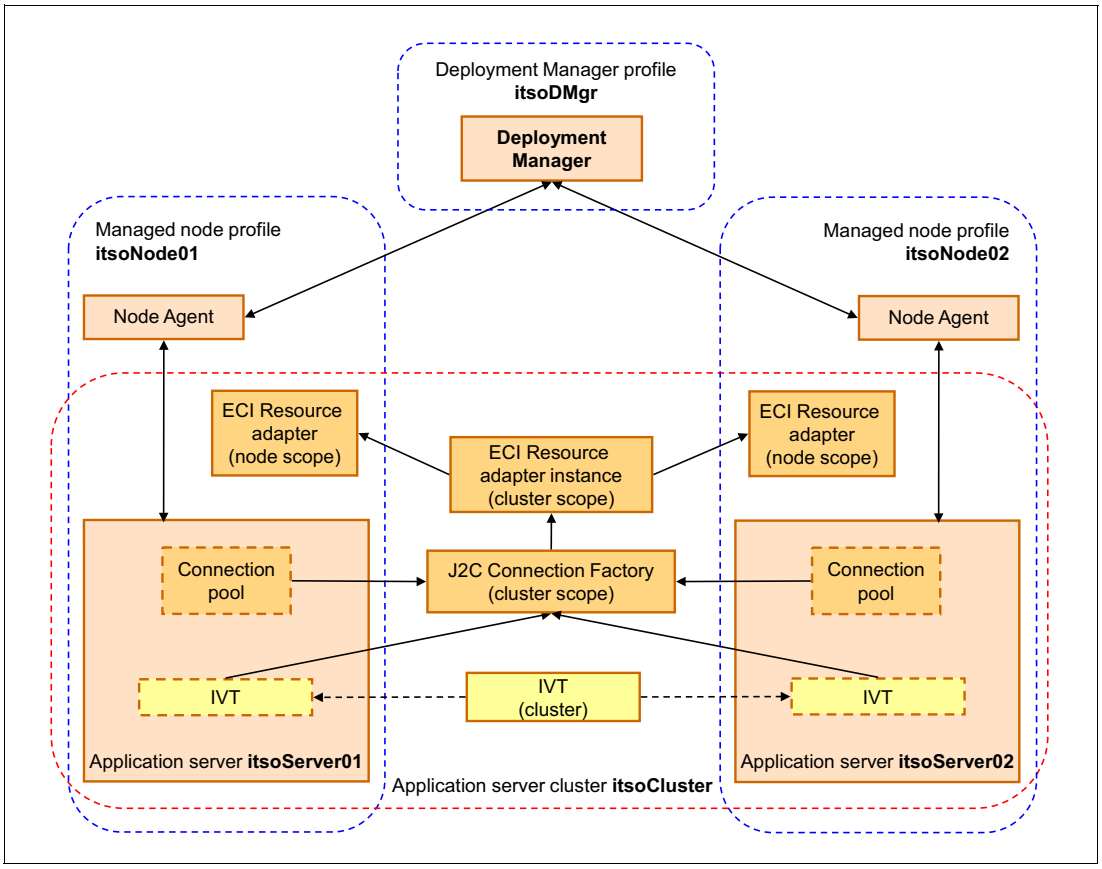


Figure 13-5 WebSphere Application Server configuration on AIX

A WebSphere Application Server Network Deployment topology uses deployment managers and node agents, in addition to the application servers:

- *Deployment managers* are administrative agents that provide a centralized management view for all nodes in a cell, as well as the management of clusters and the workload balancing of application servers across one or more nodes. Each cell contains one deployment manager.
- *Node agents* are administrative agents that represent a node to your system and manage the servers on that node. Node agents monitor application servers on a host system and route administrative requests to servers.

Considerations for configuring a high availability WebSphere Application Server configuration:

- The WebSphere Application Server cluster used in this chapter consists of two application servers hosted on the same physical machine. In a production environment, it is suggested to configure the application servers and their associated managed nodes on different (physical or virtual) machines, to provide horizontal scaling. Also, scale the number of application servers to meet capacity demands.
- Java EE applications with a web interface installed in WebSphere Application Server can be invoked directly by using the application server's HTTP transport. However, for high availability scenarios, the use of a separate HTTP server, for example, IBM HTTP Server, is suggested. On Multiplatforms, the WebSphere Application Server Network Deployment product includes the IBM HTTP Server and Web Server Plugins installation files. The Web Server Plugins are used to configure the IBM HTTP Server, so that it can distribute HTTP requests to WebSphere Application Server application servers and can also be administered from the administration console as a Web Server.
- The configuration used in this chapter uses a single IBM HTTP Server to distribute requests to the WebSphere Application Server cluster. In a production environment, it is suggested to configure multiple HTTP servers on different (physical or virtual) machines, to provide horizontal scaling, with each capable of distributing requests to each of the application servers. A load balancer, such as the load balancer provided by WebSphere Edge Components, can be used to distribute requests between the HTTP servers.

For information about installing and configuring a clustered topology, see the WebSphere Application Server Information Center.

13.6.2 Using ECI resource adapter in a WebSphere Application Server cluster

This section describes additional considerations when using the CICS ECI resource adapter in a WebSphere Application Server clustered configuration.

Installing ECI resource adapter and configuring Java EE Connector Architecture connection factory

Chapter 12, "Integrating with WebSphere Application Server" on page 261 describes how to install the ECI resource adapter and configure a Java EE Connector Architecture connection factory. This section highlights the considerations when using a WebSphere Application Server cluster configuration:

- Access the administration console from the deployment manager. When saving configuration changes, ensure that the changes are synchronized to the nodes through the node agents.
- Because JCA resource adapters can only be installed at the WebSphere Application Server node scope, they must be installed to each node used by the clustered application servers.

- Also, Java EE Connector Architecture connection factories can only be created at the same scope as an installed JCA resource adapter. To avoid the need to define a Java EE Connector Architecture connection factory for each node used by a clustered application server and manually ensure that the definitions are kept identical, a better solution is to create a resource adapter instance at the cluster scope. For instructions to create a resource adapter instance, see 12.5.1, “Install the ECI resource adapter” on page 270. A single Java EE Connector Architecture connection factory can then be created at the cluster scope and shared between the clustered application servers. A separate instance of the connection pool is created in each application server. For example, for a cluster with two application servers that use the same Java EE Connector Architecture connection factory with a connection pool `Maximum Connections` property setting of `100`, you can generate up to 200 connections (two servers with 100 connections each).
- For remote mode connections, the Java EE Connector Architecture connection factory custom property `serverName` specifies the name of a CICS server defined in the Gateway daemon configuration file. However, to implement high availability for the CICS TS regions using Dynamic server selection (see 13.5, “Configuring DSSPOLICY” on page 311), use a value defined in the Gateway daemon configuration file DSSPOLICY MAPPINGS definition.

  For this high availability topology, set `serverName=CICSTOR`.
- When you install Java EE applications, which use the Java EE Connector Architecture connection factory, ensure that they are mapped to the cluster instead of individual application servers. Also, if the application contains any web components, ensure that these web components are also mapped to any Web Servers defined to the WebSphere Application Server cell.

13.6.3 Configuring a Java EE Connector Architecture connection factory for CICS TG remote mode

This section describes considerations for configuring a Java EE Connector Architecture connection factory in a clustered topology.

Table 13-2 displays the Java EE Connector Architecture connection factory custom properties used in this configuration.

Table 13-2 Java EE Connector Architecture connection factory custom properties used in this configuration

Property name	Property value	Description
connectionURL	`tcp://lp00.redbooks.ibm.com`	Remote mode connection to the Gateway daemons `tcp` protocol handler. This uses the host name of the DVIPA for the sysplex.
port	`2006`	The `tcp` protocol handler port defined for the Gateway daemons. Defined as a TCP/IP shared port and added to Sysplex Distributor port list.
serverName	`CICSTOR`	Values specified in the Gateway daemon `DSSPOLICY` section `MAPPINGS` property. Allow DSS to determine the CICS TS region to use.
xaSupport	`off`	Enable XA transaction support. See 13.7, “XA considerations” on page 318.

Table 13-3 on page 318 displays the connection pool settings for the Java EE Connector Architecture connection factory used in this configuration.

Table 13-3 Connection pool settings used in this configuration

Property name	Property value	Description
Connection timeout	60 secs	If connection cannot be obtained from the pool within 60 seconds, time out the request.
Maximum connections	100	Because we have two application servers in the cluster, we have a maximum of 200 connections (matching the number of users).
Minimum connections	50	Because we have two application servers in the cluster, this effectively means that a minimum of 100 connections are maintained in the connection pool, which matches the expected number of concurrent requests. Increasing from a default of 1 reduces delays in obtaining connections after periods of low activity or the application servers being restarted.
Reap time	180 secs	The default value
Unused timeout	600 seconds	Remove unused connections after 3 minutes, which can be useful to ensure connections are established to Gateway daemons that have previously been removed from the TCP/IP workload balancing distribution. When a Gateway daemon health value falls to zero, WLM will cause Sysplex Distributor and TCP/IP port sharing to no longer distribute requests to it. However, when it recovers, all the connections in the connection pool will be connecting to other Gateway daemons. While connections exist in the pool, they will be used and no attempts will be made to establish connections to currently available Gateway daemons.
Aged timeout	600 seconds	See unused timeout.
Purge policy	Entire pool	When a stale connection or fatal connection error is detected, all connections in the pool are marked stale. A connection in use is closed and issues a stale connection exception during the next operation on that connection. Subsequent getConnection() requests from the application result in new connections opening.

Avoiding problems using local mode IPIC connections

When you use a WebSphere Application Server cluster configuration with multiple application servers, problems can occur using either auto-installed connections that use the CICS default IPIC autoinstall user program `DFHISAIP` or using predefined IPIC connections to connect to CICS TS. This issue is described in more detail in 4.4, “IPIC autoinstall for WebSphere Application Server clusters” on page 64. The steps to implement the solution used in this book are described in 9.5, “Configuring IPIC autoinstall for WebSphere Application Server clusters” on page 167.

Additional considerations for WebSphere Application Server on z/OS

If you use WebSphere Application Server for z/OS, an additional high availability configuration option is available. The WebSphere Application Server cluster can be configured to work with Sysplex Distributor to provide either workload balancing or failover for incoming requests from a browser or client application.

13.7 XA considerations

This section describes additional considerations to implement global transactions in a high availability environment.

To allow each of the Gateway daemons to recover XA transactions on behalf of any of the other Gateway daemons, you must permit access for the Gateway daemon user ID to one of the appropriate MVSADMIN.RRS.COMMANDS RACF facilities when XA support is enabled. For more details, see the “Configuring for XA transaction support” topic in the CICS TG Information Center.

In the high availability environment shown in Figure 13-1 on page 297, we chose to permit the Gateway daemons to perform recovery operations for transactions associated with any system name or logging group. This is achieved by permitting the Gateway daemon user ID `ALTER` access to the MVSADMIN.RRS.COMMANDS.** facility.

To run the `IVT` Java EE application to use global transactions, set the Java EE Connector Architecture connection factory custom property `xaSupport=on` and restart the WebSphere Application Server cluster for the change to take effect. The `IVT` EJB specifies Transaction=Required in its deployment descriptor so it will initiate the global transaction.

For more details about the concepts for implementing XA in a high availability environment, see 7.5, “Considerations for XA transaction support on z/OS” on page 124.

For more information about configuring XA support in z/OS Gateway daemons, see 10.5, “Adding XA support” on page 198.

14

Explorer plug-in

This chapter provides an overview of installing IBM CICS Transaction Gateway plug-in for CICS Explorer into IBM Explorer for z/OS, and using it to configure, start, and stop a z/OS Gateway daemon. It also describes the CICS Transaction Gateway perspective, explaining how to view status and activity information for Gateway daemons on all platforms.

This chapter explains how to configure a z/OS Gateway daemon using IBM Explorer for z/OS, use CICS TG plug-in for CICS Explorer to connect to Gateway daemons, and navigate the CICS Transaction Gateway perspective.

This chapter contains the following topics:

14.1 Introduction

IBM Explorer for z/OS (z/OS Explorer) and IBM CICS Explorer both provide a common, intuitive, Eclipse-based rich client platform for architects, developers, programmers, and administrators. These products contain powerful task-oriented views, context-sensitive resource editors, and wizards to provide integrated access to a broad range of CICS and z/OS data and control capabilities. Both products are extendable through the IBM repository of compatible products, including IBM CICS Transaction Gateway plug-in for CICS Explorer, to fulfill each user's roles and responsibilities.

z/OS Explorer provides a common connection management and single sign-on component to simplify access to z/OS-based subsystems, and an intuitive and secure way to view, edit, and manage z/OS data sets and z/OS file system (zFS) files, submit JCL, and view JES output and job logs.

CICS Explorer integrates the visibility and control of the CICS run time and its resource definitions within a single, consistent user interface. A wide range of IBM CICS and Problem Determination tools can be installed into CICS Explorer from the IBM web-based repository. This helps both novice and experienced CICS users get more from CICS, and accelerates the transfer of knowledge, skills, and preferred practices to the next generation.

You can use either of the following common approaches to obtain the capabilities of z/OS Explorer, CICS Explorer, and CICS TG plug-in for CICS Explorer:

- Install z/OS Explorer with CICS Explorer software development kit (SDK) plug-in and CICS TG plug-in for CICS Explorer.
- Install CICS Explorer, which embeds z/OS Explorer, and then install CICS TG plug-in for CICS Explorer.

Note: z/OS Explorer is also embedded within other IBM products, including IBM Rational Developer for System z. For more information about the installation options and capabilities of these products, see these websites:

- z/OS Explorer: `http://www.ibm.com/software/htp/cics/ibmexplforzos/`
- CICS Explorer: `http://www.ibm.com/software/products/us/en/cics-explorer`

CICS TG plug-in for CICS Explorer

CICS TG plug-in for CICS Explorer can be used in either z/OS Explorer or CICS Explorer. It allows users to view status and activity for Gateway daemons on both Multiplatforms and z/OS by using the statistics API. This includes information for the connections to the CICS servers from each Gateway daemon.

The examples in this chapter use z/OS Explorer, but the same functionality is available in CICS Explorer.

14.2 Installing CICS TG plug-in for CICS Explorer

This section describes how to obtain and install z/OS Explorer with the CICS Explorer SDK plug-in and the IBM CICS Transaction Gateway plug-in for CICS Explorer.

z/OS Explorer can be installed by using either IBM Installation Manager (IM), which is the suggested method, or by using the Eclipse P2 software update method. IM installs the Explorer product code and plug-ins from a network repository.

We selected the following Explorer components:

- z/OS Explorer Version 2.1.0.0
- CICS Explorer SDK Version 5.1.1.0
- IBM CICS Transaction Gateway plug-in for CICS Explorer 2.0.1.0

To install z/OS Explorer with the CICS Explorer and CICS Transaction Gateway plug-ins using IM, perform the following steps:

1. Download z/OS Explorer from the IBM download site:

   http://www.ibm.com/software/htp/cics/ibmexplforzos/downloads.html

2. Follow the installation instructions for “Scenario 1 - Install z/OS Explorer and Installation Manager” from the download site, noting the following points:

   a. From the Install Packages “Select packages to install” dialog box, select the following components:

      - **z/OS Explorer**
      - **CICS Explorer SDK**
      - **IBM CICS Transaction Gateway plug-in for CICS Explorer**

   b. Click **Cancel** when prompted for authentication with an IBM ID and jazz.net ID, because this authentication is not required for the selected packages. See the z/OS Explorer installation instructions for more details about these authentication prompts.

   c. From the Install Packages “Review the summary information” dialog box, confirm that the chosen packages and any subcomponents are selected.

   d. Wait for the installation to complete and ensure that it completes successfully.

3. Launch z/OS Explorer:

   The Welcome page opens and contains a First Steps section that explains how to define connections to systems in z/OS Explorer. It also contains tutorials for the IBM CICS Transaction Gateway plug-in and CICS Explorer.

14.3 Creating and administering z/OS Gateway daemons

This section describes how to use z/OS Explorer to create, start, and stop a z/OS Gateway daemon. It includes information about creating the z/OS partitioned data set (PDS) members and UNIX System Services files, and submitting z/OS jobs.

Table 14-1 summarizes the values that are used to configure the new Gateway daemon.

Table 14-1 Values used to configure new Gateway daemon

Gateway daemon	Attribute	Value
z/OS Test Gateway daemon	Jobname	`CTGTJOB`
z/OS Test Gateway daemon	JCL	`‘CTGUSER.CTGITSO(CTGTJOB)’`
z/OS Test Gateway daemon	Environment variables	`‘CTGUSER.CTGITSO(CTGTENV)’`
z/OS Test Gateway daemon	Configuration file	`/u/cicstguser/ctg/ctg.ini`
z/OS Test Gateway daemon	Hostname	`lp01.redbooks.ibm.com`
z/OS Test Gateway daemon	Statistics API handler port	`2989`

Gateway daemon	Attribute	Value
z/OS Test Gateway daemon	`tcp` protocol handler port	`2009`
z/OS Test Gateway daemon	SECTION IPICSERVER	`CICSTOR1`
z/OS Test Gateway daemon	APPLID/APPLIDQUALIFIER	`CTGAPP/CTGAPPQ`
CICS Transaction Server for z/OS (CICS TS)	IPIC TCPIPSERVICE port	`9011`
CICS TS	IPIC TCPIPSERVICE hostname	`lp01.redbooks.ibm.com`
z/OS FTP	hostname and port	`lp99.redbooks.ibm.com` `21`
z/OSMF (z/OS Management Facility)	hostname and port	`lp99.redbooks.ibm.com` `32208`
z/OS credentials	User ID/password	`CTGUSER/password`

14.3.1 Creating a z/OS Gateway daemon using z/OS Explorer

This section describes how to create the CICS TG configuration and environment files and JCL to start and stop a z/OS Gateway daemon using z/OS Explorer. For detailed information about configuring a z/OS Gateway daemon, see 10.1, “Configure the Gateway daemon” on page 172.

The CICS TG for z/OS configuration consists of two parts:

- A set of environment variables. The environment variables are stored as a PDS member. A sample environment variables PDS member is in `CTG.V9R0M0.SCTGSAMP`(CTGENV).
- A text-based configuration file. The configuration file can be stored as either a PDS member or as a UNIX System Services file. Sample configuration files are in PDS member `CTG.V9R0M0.SCTGSAMP`(CTGCONF) and `/usr/lpp/cicstg/ctg900/samples/configuration/ctgsamp.ini`.

  Note: The ability to store the configuration file in a PDS was introduced in CICS TG for z/OS V8.1.

  For this example, the configuration file will be created as a UNIX System Services file to demonstrate how the z/OS Explorer can work with UNIX System Services files and directories.

IBM for z/OS Explorer consists of various perspectives. A *perspective* defines the initial set and layout of views in the Workbench window. Each perspective provides a set of functionality aimed at accomplishing a specific type of task or works with specific types of resources. The available perspectives depend on which plug-ins have been installed. For this example, it includes the z/OS (the default), CICS Transaction Gateway, and CICS SM perspectives.

Each perspective consists of multiple views. The z/OS perspective contains the following views:

- Data sets: Lists all the data sets that you are authorized to view and that match the data set name qualifier specified. You can open a data set to view the members.
- z/OS UNIX files: Shows the zFS file system structure and contents. The files are shown in a tree structure and you can expand the tree to show individual files.

- Editor: A text editor is used to view and edit content of PDS members and UNIX System Services files.
- Jobs: Lists the completed and running jobs that you are authorized to view and that match the job name prefix and owner ID specified.
- z/OS job: Shows all the output data sets for a specified job.
- Properties: Displays all the properties, or attributes, of the selected resource.
- Console: History of the actions performed, such as submitting a job, or changing and saving a file.

Figure 14-1 shows the z/OS perspective of IBM for z/OS Explorer.

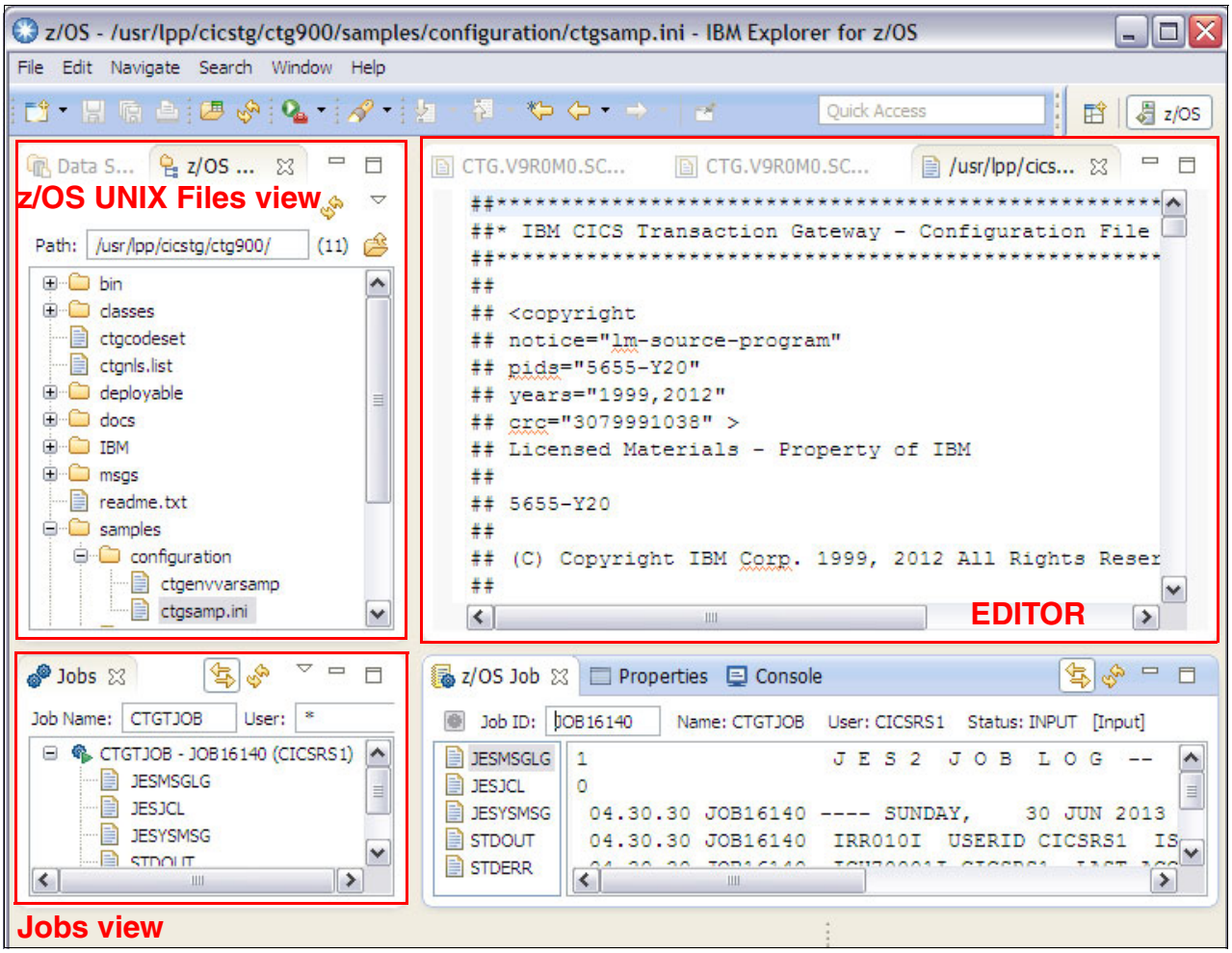


Figure 14-1 z/OS perspective of z/OS Explorer

z/OS Explorer establishes connections to z/OS using FTP and connections to the z/OS Management Facility (z/OSMF). *z/OSMF* is a management product for z/OS that provides support for a modern, web browser-based console for z/OS. It is intended to enable IT personnel, such as system programmers, administrators, and others, to more easily manage a mainframe by simplifying the configuration and management of a z/OS system. For more information about z/OSMF, see the IBM z/OS Management Facility:

http://www.ibm.com/systems/z/os/zos/zosmf/

An FTP connection is required to use the function in the z/OS perspective. A z/OSMF connection provides additional functionality: the ability to view the output of active jobs while they are running and delete or cancel active jobs. When you define a z/OSMF connection, you also define an FTP connection. z/OS Explorer then chooses the connection that is most suitable for the task that you are performing.

Use the following steps to create the configuration resources for the Gateway daemon:

1. Create a connection to z/OSMF:
   a. Select **Windows** → **Manage Connections** to open the Host Connections view as shown in Figure 14-2 on page 327.
   b. Define credentials to access z/OS. In the Credentials pane, click **Add** and complete the New credentials dialog box:
      i. Set Credentials name to `ITSO zOS credentials`.
      ii. Set User ID to `CTGUSER`.
      iii. Set Password or Passphrase to `password`.
      iv. Optional: Select **Save password**.
      v. Click **OK**.
   c. Define an FTP connection to z/OS. In the Connections pane, click **Add** → **z/OS FTP** and complete the Add z/OS FTP Connection dialog box:
      i. Set Name to `ITSO Sysplex FTP`.
      ii. Set Host name to `lp99.redbooks.ibm.com`.
      iii. Set Port to `21`.
      iv. Click **Save and Close**.
   d. Define a Secure Sockets Layer (SSL) connection to the z/OSMF. In the Connections pane, click **Add** → **z/OSMF FTP** and complete the Add z/OSMF Connection dialog box:
      i. Set Name to `ITSO Sysplex zOSMF`.
      ii. Set Host name to `lp99.redbooks.ibm.com`.
      iii. Set Port to `32208`.
      iv. Select **Secure connection**.
      v. Set FTP connection to `ITSO Sysplex FTP`.
      vi. Click **Save and Connect**.
      vii. The first time that you connect, a Sign-on prompt is displayed. Select **Use existing Credentials** and set it to `ITSO zOS credentials`. Click **OK.** Confirm the credential details and click **OK**.

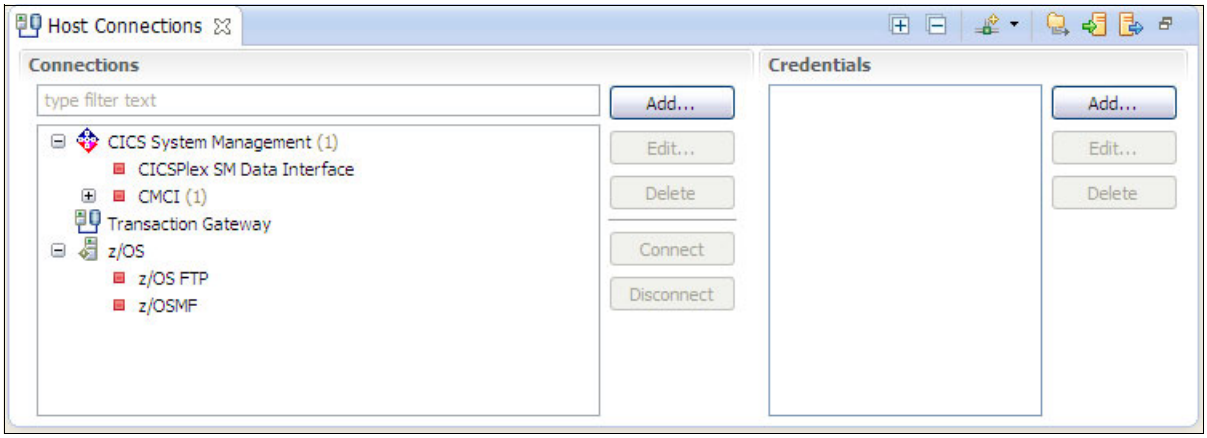


Figure 14-2 Host Connections view

2. Open the z/OS perspective:

   This is the default perspective, but if it is not currently open, open it by selecting **Window** → **Open Perspective** → **Other** → **z/OS (default)**.

3. Create a configuration file in UNIX System Services from the z/OS UNIX Files view:

   Note: UNIX System Services paths and file names are case-sensitive.

   a. Create a new UNIX System Services directory and configuration file, for example, `/u/cicstguser/ctg/ctg.ini`:

      i. Enter `/u/cicstguser` in the Path field (in this example, this directory already exists).

      ii. Right-click in the z/OS UNIX Files view and select **New Directory**.

      iii. In the New z/OS directory dialog box, enter `ctg` as the New directory name and click **Finish**.

      iv. In the z/OS UNIX Files view, open the new directory by double-clicking **ctg**.

      v. Right-click in the z/OS UNIX Files view and select **New File**.

      vi. In the New z/OS file dialog box, enter `ctg.ini` as the New file name and click **Finish**.

      vii. Double-click **ctg.ini** to open it in the editor. (The file is initially empty; leave the file open.)

   b. Copy the contents of the CICS TG sample configuration file `/usr/lpp/cicstg/ctg900/ctg/samples/configuration/ctgsamp.ini` into the new configuration file, for example, `/u/cicstguser/ctg/ctg.ini`:

      i. Enter CICS TG `/usr/lpp/cicstg/ctg900/samples/configuration` in the Path field.

      ii. Double-click **ctgsamp.ini** to open it in the editor. (Leave the file open.)

      iii. Press Ctrl+A to select all the contents of `ctgsamp.ini`. Press Ctrl+C to copy and then press Ctrl+P to paste the contents into `ctg.ini`. Press Ctrl+S to save the new `ctg.ini` file.

      iv. Close the `ctgsamp.ini` file.

      Note: As an alternative to using the keyboard shortcuts, use the File and Edit menus from the z/OS Explorer toolbar.

c. Customize the configuration file to enable the `tcp` protocol and statistics API handlers and define an `IPICSERVER` definition to a CICS TS region. Edit the following values in the configuration file `/u/cicstguser/ctg/ctg.ini` in the editor:

   i. In the `PRODUCT` section, uncomment the `SECTION` and `ENDSECTION` statements.

   ii. In the `PRODUCT` section: uncomment and set `APPLID` to `CTGAPP` and set `APPLIDQUALIFIER` to `CTGAPPQ`.

   iii. In the `GATEWAY` section, uncomment the `SECTION` and `ENDSECTION` statements.

   iv. In the `GATEWAY` section, uncomment the Protocol handler for statistics API requests definitions and change the port to a unique value, for example, `2989`.

   v. In the `GATEWAY` section, uncomment the Protocol handler for TCP/IP requests definitions and change the port to a unique value, for example, `2009`.

   vi. In the `IPICSERVER` section, uncomment the `SECTION` and `ENDSECTION` statements.

   vii. In the `IPICSERVER` section, change the server name to `CICSTOR1`.

   viii. In the `IPICSERVER` section, uncomment and set the Hostname to `lp01.redbooks.ibm.com` and set the Port to `9011`.

   ix. Save the file by pressing Ctrl+S.

   x. Check the attributes of the configuration file by right-clicking the file name, for example, `ctg.ini`, and selecting **Properties**. The user ID under which the Gateway daemon runs requires at least read access to the file.

   Example 14-1 shows the properties specified in the `ctg.ini` file. (The other properties were left to use the default values.) For information about other configuration properties, see 10.1.3, “Configuration file” on page 174.

Example 14-1 The ctg.ini file properties

```
SECTION PRODUCT
   Applid = CTGAPP           # The APPLID of the Gateway daemon
   ApplidQualifier = CTGAPPQ    # The APPLID qualifier of the Gateway daemon
ENDSECTION

SECTION GATEWAY
   # Protocol handler for statistics API requests
   protocol@statsapi.handler = com.ibm.ctg.server.RestrictedTCPHandler
   protocol@statsapi.parameters = bind=;\
                              connecttimeout=2000;\
                              maxconn=5;\
                              port=2989;

   # Protocol handler for TCP/IP requests
   protocol@tcp.handler = com.ibm.ctg.server.TCPHandler
   protocol@tcp.parameters = bind=;\
                          connecttimeout=2000;\
                          idletimeout=600000;\
                          pingfrequency=60000;\
                          port=2009;\
                          solinger=10;
ENDSECTION
```

```
SECTION IPICSERVER = CICSTOR1  # Arbitrary name for the server
   Hostname = lp01.redbooks.ibm.com # The server's TCP/IP name or IPv4 or
IPv6 address
   Port = 9011                  # The listening TCPIPSERVICE port in CICS
ENDSECTION
```

4. On the Data Sets view, create an environment variables file as a PDS member in a new PDS:
   a. To create the PDS that will contain the environment variables file and JCL to start and stop the z/OS Gateway daemon, follow these steps:
      i. Right-click in the Data Sets view, and select **New Source or JCL Data Set**, which opens the New Data Set dialog box.
      ii. Set Data Set Name to `CTGUSER.CTGITSO`.
      iii. Accept the default values for other properties, including the Type of Library, Record Format = `Fixed Blocked`, and Record Length = `80`.
      iv. Click **Finish**.
   b. To create a new member in the PDS, right-click the PDS name, for example, `CTGUSER.CTGITSO`, in the Data Sets view, and select **New Data Set Member**. This opens the New Data Set Member dialog box. Enter the following values:
      i. Set Partition Data Set Name to `CTGUSER.CTGITSO`.
      ii. Set Member Name to `CTGTENV`.
      iii. Select **Open Editor**.
      iv. Click **Finish**.

      The empty `CTGTENV` is opened in the editor. (Leave the file open.)
   c. Copy the contents of the sample environment variable PDS member from the CICS TG installation PDS `CTG.V9R0M0.SCTGSAMP`(CTGENV) to the new PDS `CTGUSER.CTGITSO`:
      i. Enter `CTG.V9R0M0.SCTGSAMP` in the Qualifier field.
      ii. Double-click **CTGENV** to open it in the editor.
      iii. Press Ctrl+A to select all contents. Press Ctrl+C to copy, and then press Ctrl+P to paste the contents into `CTGUSER.CTGITSO(CTGTENV)`. Press Ctrl+S to save the new `CTGTENV` file.
   d. Customize the environment variable PDS member `CTGTENV` in the editor:
      i. Set `CICSCLI=/u/cicstguser/ctg/ctg.ini` (location of the configuration file).
      ii. Set `PATH=/bin:/usr/lpp/java/J7.0/bin`.
      iii. Comment `STEPLIB`, `DFHJVSYSTEM_OO`, and `CTG_PIPE_REUSE`. These are not required because this example does not use external CICS interface (EXCI) connections.
      iv. Uncomment statement `CTG_EXCI_INIT=NO`, because this example does not use EXCI connections. This allows our z/OS Gateway daemon IPIC connections to use more than 250 connections.
      v. Accept the default values for the other parameters. For information about other environment variables, see 10.1.4, “Environment variables” on page 176.
      vi. Save the file by pressing Ctrl+S.

   Example 14-2 on page 330 shows the contents of the **`CTGTENV`** environment variable PDS member.

Example 14-2 CTGTENV file contents

```
CICSCLI=/u/cicstguser/ctg/ctg.ini
PATH=/bin:/usr/lpp/java/J7.0/bin
_BPX_SHAREAS=YES
CTG_EXCI_INIT=NO
AUTH_USERID_PASSWORD=NO
COLUMNS=80
```

14.3.2 Starting and stopping a z/OS Gateway daemon

z/OS Explorer can be used to submit z/OS jobs and view the job output. This section describes how to stop and start a z/OS Gateway daemon by submitting z/OS jobs.

Starting a z/OS Gateway daemon

A z/OS Gateway daemon address space can be started as either a job or procedure, both requiring JCL stored as a PDS member. This example starts the Gateway daemon using a job, based on the sample PDS member provided in `CTG.V9R0M0.SCTGSAMP`(CTGJOB).

Create JCL to start the Gateway daemon as a job in a PDS member of an existing PDS:

1. From the Data Sets view, enter `CTGUSER.CTGITSO` in the Qualifier field.
2. To create a new member in the PDS, right-click the PDS name, for example, `CTGUSER.CTGITSO`, in Data Sets view and select **New Data Set Member**. This opens the New Data Set Member dialog box.
3. In the New Data Set Member dialog box, enter the following values:
   - Set Partition Data Set Name to `CTGUSER.CTGITSO`.
   - Set Member Name to `CTGTJOB`.
   - Select **Open Editor**.
   - Click **Finish**.

   The empty `CTGTJOB` is opened in the editor. (Leave the file open.)
4. Copy the contents of the sample JCL to start the Gateway daemon PDS member from the CICS TG installation PDS `CTG.V9R0M0.SCTGSAMP`(CTGJOB) to the new PDS `CTGUSER.CTGITSO`:
   a. Enter `CTG.V9R0M0.SCTGSAMP` in the Qualifier field.
   b. Double-click **CTGJOB** to open it in the editor.
   c. Press Ctrl+A to select all the contents. Press Ctrl+C to copy and press Ctrl+P to paste the contents into `CTGUSER.CTGITSO(CTGTJOB)`. Press Ctrl+S to save the new `CTGTJOB` file.
5. Customize the environment variable PDS member **`CTGTJOB`** in the editor:
   a. Change the job name to `CTGTJOB`.
   b. Add a `JOBPARM SYSAFF` statement to indicate the LPAR on which the address space will run, for example, `/*JOBPARM SYSAFF=LP01`.
   c. Change <*`Install path`*> to the fully qualified path of the CICS TG UNIX System Services installation directory, for example, `/usr/lpp/cicstg/ctg900`.
   d. Change <*`Install hlq`*> to the high-level qualifier (HLQ) of the CICS TG installation, for example, `CTG.V9R0M0`.

e. Change <STDENV DSN> to the environment variable PDS member, for example, `CTGUSER.CTGITSO(CTGTENV)`.

f. Press Ctrl+S to save your changes. You might receive a prompt informing you that the file changed in the host and asking whether you want to overwrite the file. Click **Save**.

Example 14-3 shows the contents of the JCL to start the Gateway daemon. Ensure that the values on the `SET CTGHOME`, `CTGHLQ`, `CTGUSR`, and `LEOPTS` statements are enclosed in single quotation marks.

Example 14-3 JCL to start Gateway daemon

```
//CTGTJOB  JOB (0),MSGCLASS=X,CLASS=A,NOTIFY=&SYSUID,REGION=500M
/*JOBPARM SYSAFF=LP01
// SET CTGHOME='/usr/lpp/cicstg/ctg900'
// SET CTGHLQ='CTG.V9ROM0'
// SET CTGUSR='CTGUSER.CTGITSO(CTGTENV)'
// SET LEOPTS='/'
//CTG      EXEC PGM=CTGBATCH,
//         PARM='&LEOPTS.&CTGHOME./bin/ctgstart -noinput '
//STEPLIB  DD DSN=&CTGHLQ..SCTGLOAD,DISP=SHR
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDENV   DD DSN=&CTGUSR.,DISP=SHR
//
```

Use these steps to start the Gateway daemon from the z/OS perspective:

1. Submit the '`CTGUSER.CTGITSO(CTGTJOB)`' job:
   a. From the Data Sets view, navigate to '**CTGUSER.CTGITSO**'.
   b. Right-click the PDS member **CTGTJOB** and select **Submit z/OS Job**.
   c. From the Perform Operation dialog box, confirm that the correct job was selected and click **OK**.
2. Check that the Gateway daemon job is running:
   a. From the Jobs view, set the filter Job name to `CTGTJOB` and the filter User to *. Then, click the **Refresh** icon.
   b. Expand **CTGTJOB** to see the job log. Either double-click each file to open it in the editor or view the log steams in the z/OS Job view.
   c. The JESMSGLG contains the following message: `CTG6512I` *`CTGAPP`* `CICS Transaction Gateway initialization complete`.

      `CTGAPP` is the `APPLID` value specified in the `PRODUCT` section of Gateway daemon configuration file.

Stopping a z/OS Gateway daemon

A z/OS Gateway daemon can be stopped by issuing a modify command to the address space, for example, **`/F jobname,APPL=SHUT`**, for a normal shutdown. z/OS Explorer does not yet provide the function to issue a modify command directly, but it can submit a job to issue the command.

Use the following steps to create a job to issue the z/OS Gateway daemon shutdown command:

1. From the Data Sets view, enter `CTGUSER.CTGITSO` in the Qualifier field.
2. Right-click the PDS name, for example, `CTGUSER.CTGITSO`, in the Data Sets view and select **New Data Set Member**.
3. In the New Data Set Member dialog box, enter the following values:
   - Set Partition Data Set Name to `CTGUSER.CTGITSO`.
   - Set Member Name to `CTGTSTOP`.
   - Select **Open Editor**.
   - Click **Finish**.

   The empty `CTGTSTOP` is opened in the editor.
4. In the editor view, paste the text that is shown in Example 14-4.

   Example 14-4 JCL to stop a z/OS Gateway daemon

   ```
   //CTGTSTOP JOB (0),CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID,REGION=500M
   /*JOBPARM SYSAFF=LP01
   //STOP EXEC PGM=IEFBR14
   // F CTGTJOB,APPL=SHUT
   //
   ```

   This example will run on LPAR `LP01` and attempt to shut down the z/OS Gateway daemon address space with the jobname `CTGTJOB`.

Use the following steps to submit a job to issue the z/OS Gateway daemon shutdown command from the z/OS perspective:

1. Submit the '`CTGUSER.CTGITSO(CTGTSTOP)`' job:
   a. From the Data Sets view, navigate to '**CTGUSER.CTGITSO**'.
   b. Right-click PDS member **CTGTSTOP** and select **Submit z/OS Job**.
   c. From the Perform Operation dialog box, confirm that the correct job was selected and click **OK**.
2. Check that the job ran successfully and that the Gateway daemon shut down:
   a. From the Jobs view, set the filter Job name to `CTGTSTOP` and the filter User to *. Then, click the **Refresh** icon.
   b. Expand **CTGTSTOP** to see the job log. Either double-click each file to open it in the editor or view the log steams in the z/OS Job view. The JESYSMSG log contains the following message: `CTGTSTOP STOP - STEP WAS EXECUTED - COND CODE 0000`.
   c. From the "Jobs" view, set the filter Job name to `CTGTJOB` and the filter User to *. Then, click the **Refresh** icon to see the Gateway daemon address space job.
   d. Expand **CTGTJOB** to see the job log. Either double-click each file to open it in the editor or view the log steams in the z/OS Job view. The JESMSGLG contains the following message: `CTG6511I` *`CTGAPP`* `GATEWAY DAEMON HAS SHUT DOWN` where *`CTGAPP`* is the `APPLID` value specified in the `PRODUCT` section of the Gateway daemon configuration file.

14.4 Preparing a Gateway daemon to use Explorer

The CICS TG statistics API allows remote users to obtain live statistical data from a Gateway daemon at run time. The CICS TG plug-in for CICS Explorer uses this facility to retrieve information about the Gateway daemon status.

The CICS TG plug-in for CICS Explorer connects to an active Gateway daemon using the Gateway daemon statistics API protocol handler port. To view the details of a Gateway daemon using the plug-in, ensure that the statistics API protocol handler port has been enabled. It is not enabled, by default.

Configure the statistics API handler

To define a statistics API handler definition in the configuration file, use the following steps:

1. Edit the Gateway daemon configuration file.
2. In the SECTION `GATEWAY`, add the following entry. (The sample configuration file `ctgsamp.ini` contains a commented out version of this entry.)

   Example 14-5 Statistics API requests protocol handler definition

   ```
   # Protocol handler for statistics API requests
   protocol@statsapi.handler = com.ibm.ctg.server.RestrictedTCPHandler
   protocol@statsapi.parameters = bind=;\
                              connecttimeout=2000;\
                              maxconn=5;\
                              port=2989;
   ```

   This example uses the default port value `2989`. If required, modify this value to ensure that it is unique on the target system.
3. Restart the Gateway daemon.

14.5 Connecting to a Gateway daemon

This section describes how to configure a connection to a Gateway daemon. These instructions demonstrate how to create a new connection using the Host Connections view dialog box:

Use the following steps to add a connection for a Gateway daemon to z/OS Explorer:

1. Select **Windows** → **Manage Connections** to open the Host Connections view.
2. In the Connections pane, click **Add** → **CICS Transaction Gateway** as shown in Figure 14-3 on page 334.

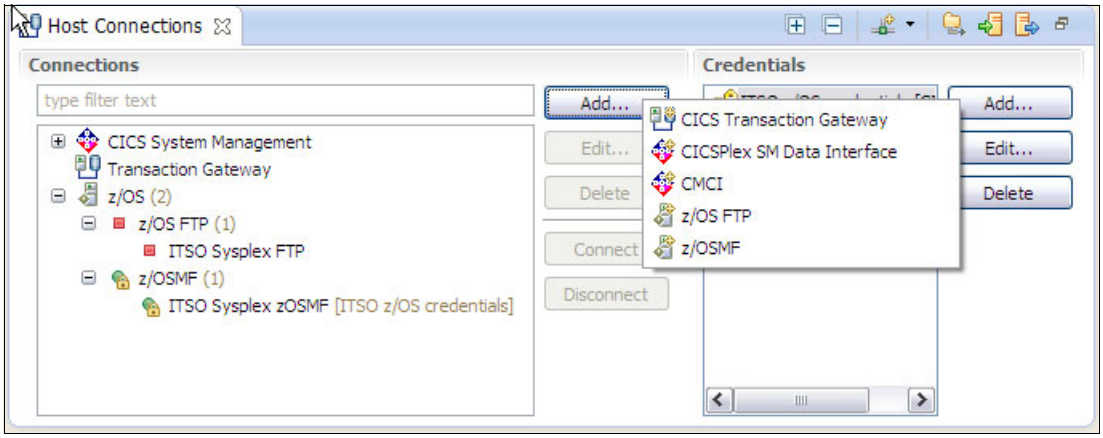


Figure 14-3 Host Connections dialog box to add CICS Transaction Gateway

3. Add a connection to the z/OS Gateway daemon configured in 14.3.1, “Creating a z/OS Gateway daemon using z/OS Explorer” on page 324. From the Add CICS Transaction Gateway Connection dialog box that is shown in Figure 14-4, enter the following values:
   - Set Name to `zOS Test Gateway`.
   - Set Host name to `lp01.redbooks.ibm.com`.
   - Set Port number to `2989` (statistics API handler port).
   - Click **Save and Connect**.

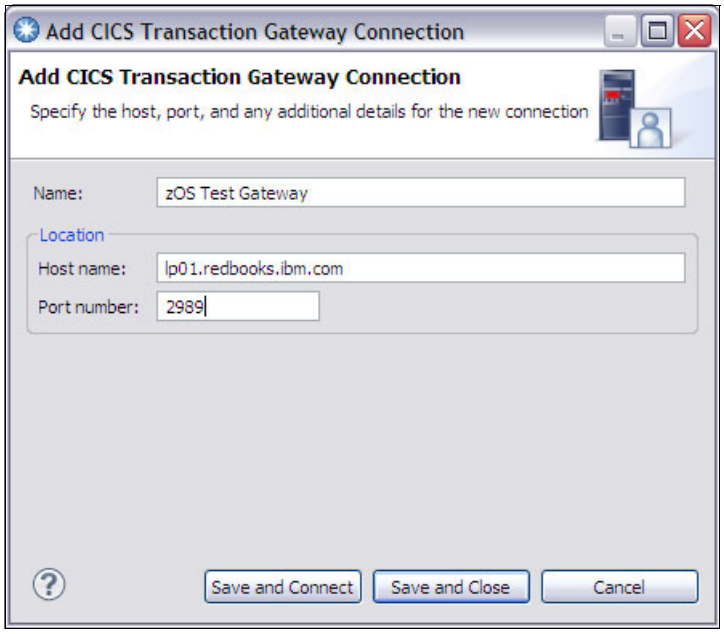


Figure 14-4 Add CICS Transaction Gateway Connection

   - The connection `zOS Test Gateway` appears in the Connections pane under the Transaction Gateway heading as shown in Figure 14-5 on page 335.

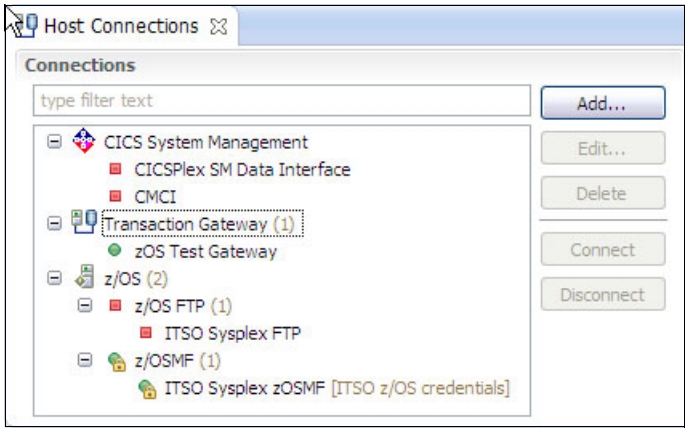


Figure 14-5 Host Connections view displaying a Gateway daemon connection

4. Repeat the steps for each Gateway daemon to which you want to connect, for example, the Gateway daemons configured in Chapter 10, “CICS Transaction Gateway for z/OS configuration” on page 171, Chapter 11, “CICS Transaction Gateway Multiplatforms and Desktop Edition configuration” on page 205, and Chapter 13, “High availability configuration” on page 295.

As an alternative to manually defining host connections on each instance of z/OS Explorer or CICS Explorer, definitions can be shared. If your systems administrator has already created a shared host connections definitions file, this file can be loaded or imported. For more information, see “Export and import of Gateway daemon, CICS TS, and z/OS connections” on page 336.

14.5.1 Backing up and restoring connection definitions

This section describes the options provided by the CICS TG plug-in for CICS Explorer and z/OS Explorer for importing and exporting connection definitions.

Export and import of connection groups and connection links

This information relates to the custom connection group and associated connection link definitions created in the custom option of the CICS TG Explorer view.

If you uninstall the CICS TG plug-in for CICS Explorer, all custom connection group and associated connection link definitions are deleted. Two options are available to avoid losing these definitions:

- Export all custom group definitions before uninstalling the CICS TG plug-in for CICS Explorer and import the definitions after installing the new version of the plug-in.
- Use the update feature when installing a new version of the CICS TG plug-in for CICS Explorer to preserve custom group definitions.

Use the following steps from the CICS TG Explorer view to export a connections group and the associated connection link definition:

1. Click the **View Menu** (triangle) icon on the CICS TG Explorer view toolbar and select **Custom**.
2. Select the connection groups to export, for example, `zOSGatewayGroup`. Right-click your selection and click **Export** as shown in Figure 14-6 on page 336.

3. Use the Save As dialog box to specify the target file name (with a file extension of `.ctg`) and a path, for example, `C:\temp\Explorer_GatewayConns.ctg`.
4. Click **OK**.

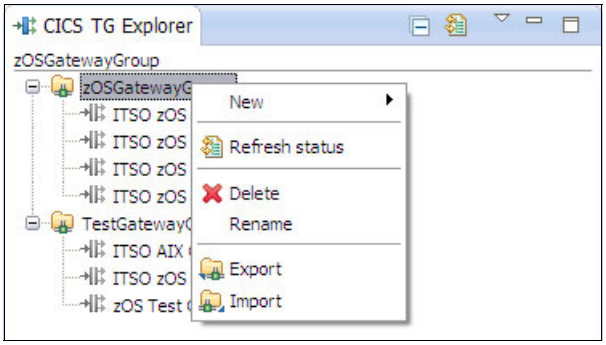


Figure 14-6 Export CICS TG custom connection group

The connection groups and the associated connection links that are previously exported can be imported to the root level or nested inside an existing connection group. Use the following steps from the CICS TG Explorer view to import the gateway connections in a connections group:

1. Click the **View Menu** (triangle) icon on the CICS TG Explorer view toolbar and select **Custom**.
2. Right-click inside the CICS TG Explorer at the root of where the imported connection groups are to be created and click **Import** as shown in Figure 14-7.
3. Use the Open dialog box to select the previously exported custom group file (file extension `.ctg`) to import, for example, `C:\temp\Explorer_GatewayConns.ctg`.
4. Click **OK**.

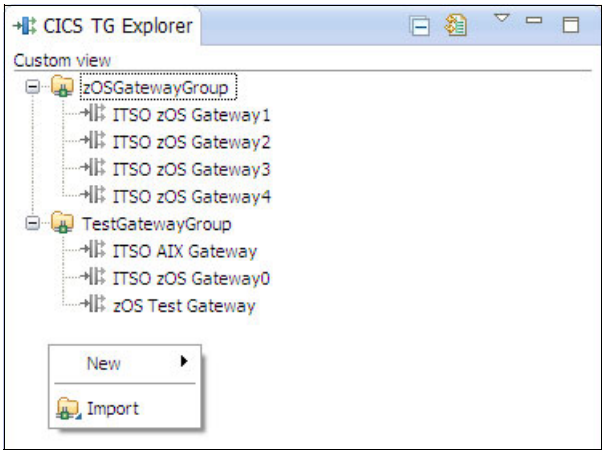


Figure 14-7 Import CICS TG custom connection group

Export and import of Gateway daemon, CICS TS, and z/OS connections

This information applies to all connection definitions created in the Host Connections view. It includes CICS TG, CICS System Management, and z/OS definitions.

As an alternative to each user manually defining their own connection definitions, a set of shared definitions can be created by a system administrator using z/OS Explorer and exported to a central location. The exported file must have a file extension of `.pref`. Any user with access to the file location can load or import the definitions in their copy of z/OS Explorer:

- When you load connections, a link is created to an external connection definition file, and the connections are shown in the Host Connections view. The definitions can be viewed but not changed. Any changes to the shared connections file are reflected the next time that z/OS Explorer starts. This has the advantage of keeping definitions synchronized across multiple workspaces, but it requires a network connection to the location of the shared definitions file. If the connection file is deleted or moved, the connections are lost from z/OS Explorer.
- When you import connections, the details are copied into your local workspace and must be manually updated if any details change. Importing a connection definition that already exists in z/OS Explorer and is associated with a credential updates only the existing connection definition details. It does not change the credential.

This approach can also be used to back up and restore the host connections in a local workspace, which have previously been created manually.

> **Note:** This approach exports all host connections currently defined in the Host Connections view and imports all host connections defined in the `.pref` file. There is not an option to select individual connections.

Use the following steps from the Host Connections view to export connection definitions:

1. Select **Windows** → **Manage Connections** to open the Host Connections view.
2. Click the **Export Connections to File** icon (blue arrow) on the Host Connections view toolbar, as shown in Figure 14-8 on page 338.
3. Use the Export Connections to File dialog box to specify the target file name (file extension of `.pref`) and path, for example, `C:\temp\Explorer_connections.pref`.
4. Click **OK**.

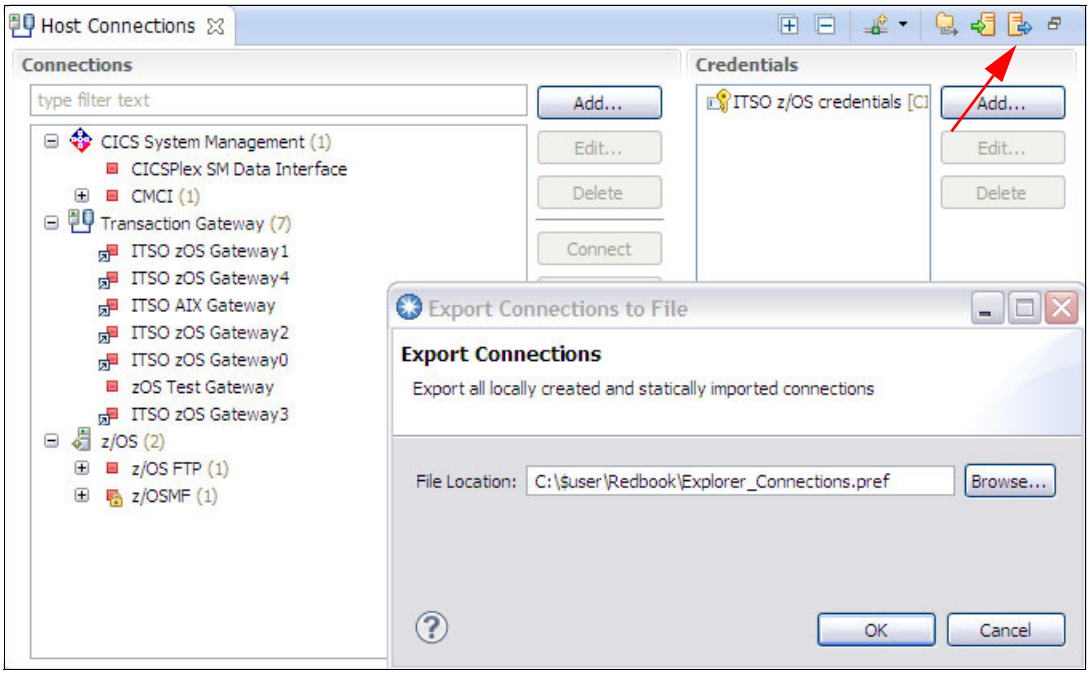


Figure 14-8 Export host connections to a file

Use the following steps from the Host Connections view to load or import connection definitions:

1. Select **Windows** → **Manage Connections** to open the Host Connections view.
2. Click **Load Connections from file or URL** icon (green arrow) on the Host Connections view toolbar, as shown in Figure 14-9 on page 339.
3. Use the Load Connections to File dialog box to select the previously exported connection definition file, for example, `C:\temp\Explorer_connections.pref`, and select the **Load** or **Import** radio button.
4. Click **OK**.

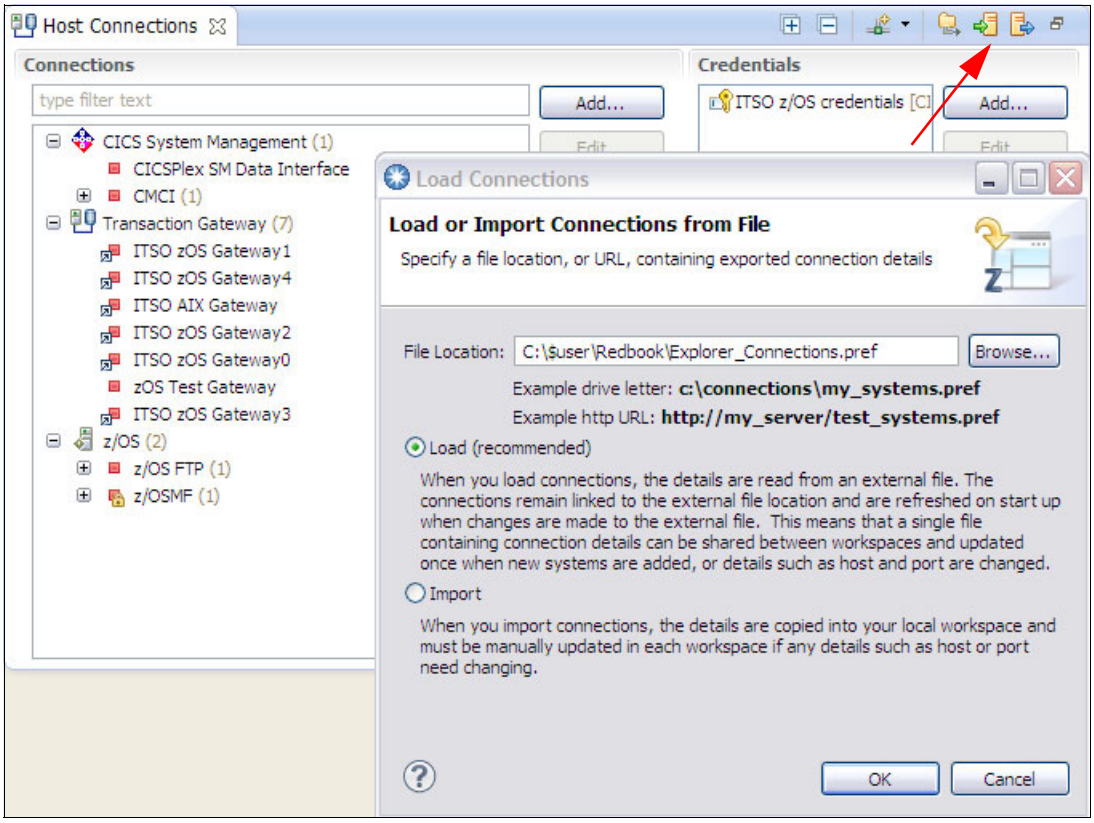


Figure 14-9 Load or import host connections

14.6 Viewing information for a Gateway daemon

This section describes how to view Gateway daemon information by using the CICS Transaction Gateway perspective. It assumes that a Gateway daemon has already been configured with at least one connection to CICS and that the Gateway daemon is running.

Open the CICS Transaction Gateway perspective by using the following steps:

1. From the z/OS Explorer toolbar, select **Windows** → **Open Perspective** → **Other**.
2. From the Open Perspective dialog box, select **CICS Transaction Gateway** and click **OK**.

The CICS Transaction Gateway perspective contains the following views, which are described in more detail in the referenced sections:

- CICS TG Explorer view: See 14.6.1, “CICS TG Explorer view” on page 340.
- Gateway daemons view: See 14.6.2, “Gateway daemons view” on page 347.
- CICS connections view: See 14.6.3, “CICS connections view” on page 348.
- Properties view
- Console view

14.6.1 CICS TG Explorer view

The CICS TG Explorer view displays the Gateway daemons and their associated CICS server connections. The Gateway daemons that are displayed are those Gateway daemons whose connections were defined using the Managed Connections view. (See 14.3, “Creating and administering z/OS Gateway daemons” on page 323.) It can also test the connections to the Gateway daemon and from the Gateway daemon to its defined CICS servers.

> **Note:** If a new Gateway daemon connection is defined and connected, but it is not displayed in this view, restart z/OS Explorer.

Sorting and organizing the view

The Gateway daemons can be displayed in various ways:

- By APPLID qualifier
- As a flattened list
- By custom connection groups

Each of these views is described in more detail:

- The Gateway daemons can be displayed as a tree view sorted by APPLID qualifier (the default option):
  - Click the **View Menu** (triangle) icon on the CICS TG Explorer view toolbar and select **APPLID qualifier**.
  - This view is useful when you use z/OS Gateway groups to see all z/OS Gateway daemons, which all share a common APPLID qualifier value, in the same group.

  Figure 14-10 displays an example APPLID qualifier view. It shows five z/OS Gateway daemons in a Gateway group (APPLID qualifier `CTGALP00`), an AIX Gateway daemon (APPLID qualifier `CTGMP`), and the z/OS test Gateway daemon created earlier in this chapter (APPLID qualifier `CTGAPPQ`).

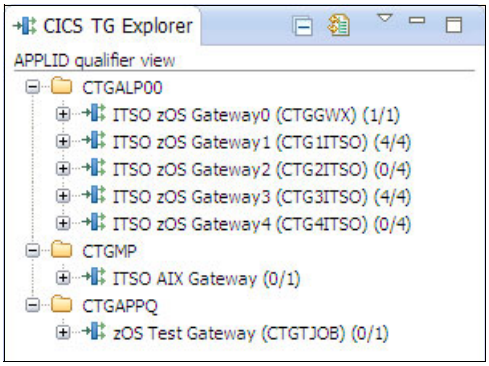


Figure 14-10 CICS TG Explorer APPLID qualifier view

- The Gateway daemons can be displayed as a flattened list that is sorted in alphabetic order:
  - Click the **View Menu** (triangle) icon on the CICS TG Explorer view toolbar and select **List**.
  - Figure 14-11 on page 341 displays an example List view. It shows an AIX Gateway daemon and six z/OS Gateway daemons, including the z/OS test Gateway daemon that was created earlier in this chapter.

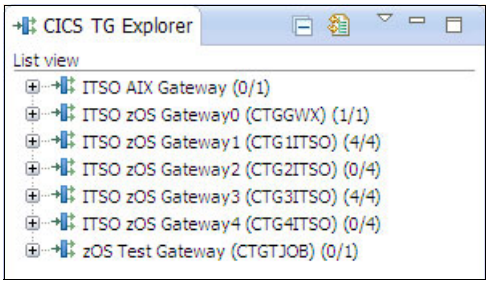


Figure 14-11 CICS TG Explorer List view

- The Gateway daemons can be displayed by using custom connection groups of selected Gateway daemons.

  The first time that you select the custom option, the CICS TG Explorer view is empty. The custom option allows custom connection groups to be created and the required Gateway daemon connections to be assigned to each group.

  Use the following steps to create a connection group and assign the Gateway daemon connections to it:

  a. Click the **View Menu** (triangle) icon on the CICS TG Explorer view toolbar and select **Custom**.

  b. Right-click inside the CICS TG Explorer view and select **New** → **Connection group**. (Alternatively, click **File** → **New** → **Connection Group** from z/OS Explorer.) The New connection group dialog is displayed.

  c. On the New connection group dialog shown in Figure 14-12 on page 342, follow these steps:

     i. Set the Connection group to **/**. (The forward slash indicates the root node.)

        This defines the position in the custom group hierarchy, the first connection group must start at the root node. Subsequent connection groups can be nested in a hierarchy. In this case, use **Browse** to select an existing connection group to use as the parent for the new group.

     ii. Set the Connection group name to `zOSGatewayGroup`.

     iii. Click **Finish**.

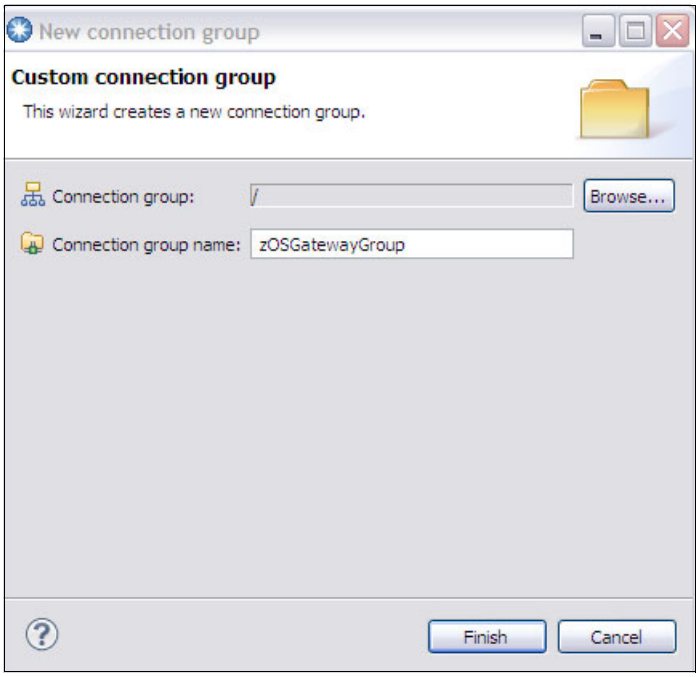


Figure 14-12 New connection group dialog

d. Right-click the connection group name, for example, `zOSGatewayGroup`, and select **New** → **Connection link**. (Alternatively, click **File** → **New** → **Gateway connection link** from z/OS Explorer.) The New connection link dialog is displayed.

e. On the New connection link dialog shown in Figure 14-13 on page 343, follow these steps:

   i. Set the Connection group name to `zOSGatewayGroup`.

   ii. From the Available connections, select the Gateway daemon connections that you want to add to the connection group from the displayed list. Multiple connections can be added to the custom group by pressing the Shift or Ctrl keys while selecting.

   iii. In this example, we chose to add four of the ITSO zOS Gateway daemons.

   iv. Click **Finish**.

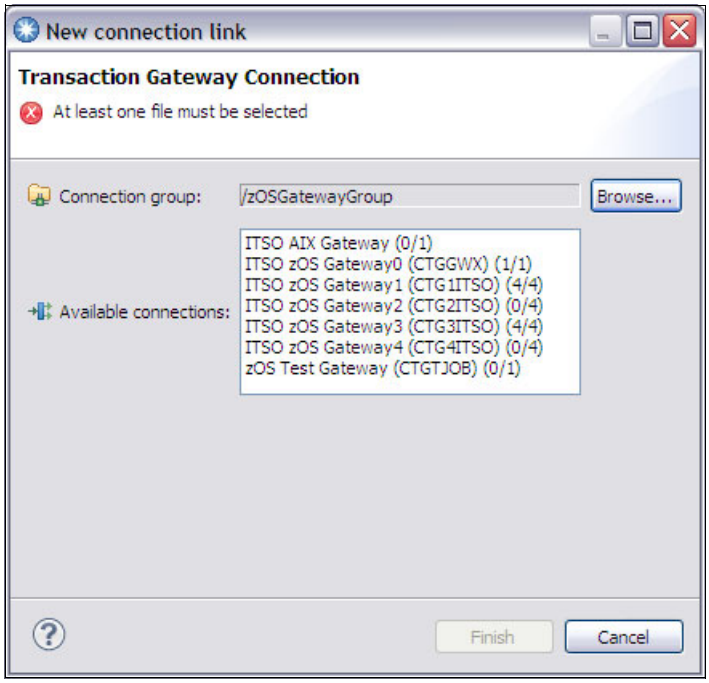


Figure 14-13 New connection link dialog

Figure 14-14 displays an example custom view. It shows a single connection group with four z/OS Gateway daemons.

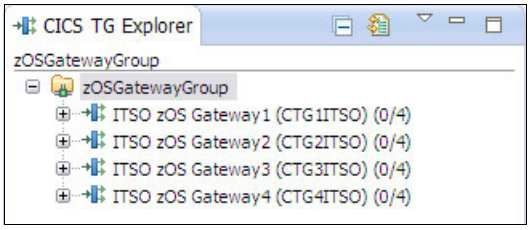


Figure 14-14 CICS TG Explorer custom view

f. Create another connection group called `TestGatewayGroup` and add the following connection links: `ITSO AIX gateway`, `ITSO zOSGateway0`, and `zOS Test Gateway`.

Testing the connection to the Gateway daemon

The CICS TG plug-in for CICS Explorer can be used to test the connection to a Gateway daemon. Follow these steps:

1. From your preferred CICS TG Explorer view option (APPLID qualifier, List, or Custom), right-click the Gateway daemon name, for example, `zOS Test Gateway`, and click **Test Gateway connection**, as shown in Figure 14-15 on page 344.

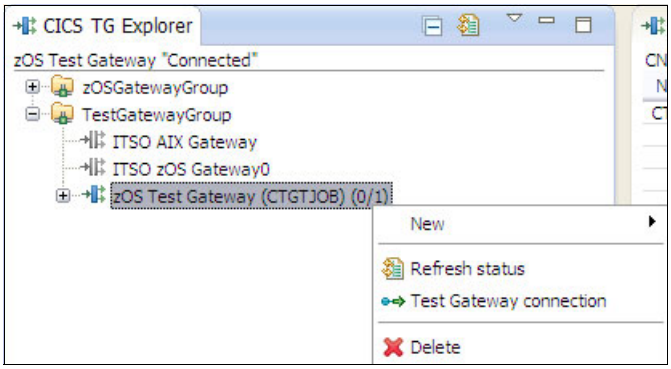


Figure 14-15 Test Gateway connection dialog

2. The result is displayed in the Console view, as shown in Example 14-6.

Example 14-6 Example output from test gateway connection

```
Attempting to open connection to Gateway daemon using host lp01.redbooks.ibm.com
on port 2,009
Success: Opened connection to lp01.redbooks.ibm.com on port 2,009
```

Testing connections to CICS servers

The CICS TG Explorer view also displays the CICS server connection definitions for each Gateway daemon.

Note: For z/OS Gateway daemons, EXCI connections to CICS servers are only displayed while they are active.

Figure 14-16 on page 345 shows an expanded version of the CICS TG Explorer view sorted by the APPLID qualifier:

- The numbers in parentheses after the Gateway daemon connection name indicate the number of active CICS connections and the number of defined CICS server connections. For example, Gateway daemon connection `ITSO zOSGateway1` displays (4/4), which indicates that the Gateway daemon configuration file contains four CICS server definitions, all of which are currently active. Also, `zOS Test Gateway` displays (0/1), which indicates that the Gateway daemon configuration file contains a single CICS server definition, which is not currently active.
- The CICS server definitions are displayed below each Gateway daemon by expanding the Gateway connection name. For example, the Gateway daemon connection `ITSO zOSGateway1` has four server definitions: `CICSTOR1`, `CICSTOR2`, `CICSTOR3`, and `CICSTOR4`. The Gateway daemon connection `zOS Test Gateway` has one server definition, `CICSTOR1`.

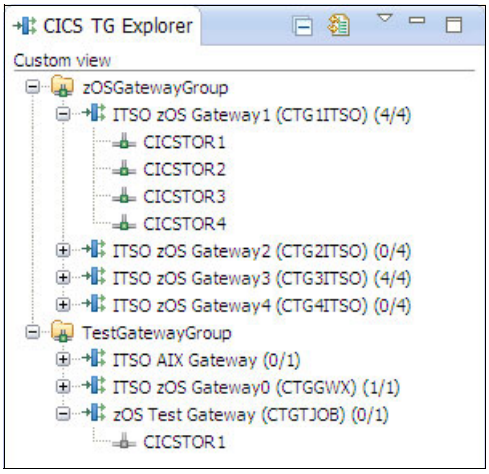


Figure 14-16 CICS TG Explorer expanded view showing CICS connections

The CICS TG plug-in for CICS Explorer can be used to test the connection to a CICS region using the Gateway daemon in remote mode. By default, the test is configured to call CICS server program `PONG` with a 32-byte COMMAREA; however, this can be customized.

Note: The `PONG` program is available from CICS SupportPac CH50 (CTGPING).

In the following example, the default configuration was customized to use the CICS TG sample server program `EC01` instead. This program requires an 18-byte COMMAREA.

Use the following steps to test an IPIC connection to CICS TS by running program `EC01`:

1. Configure z/OS Explorer to set the parameters to use to test the connection, for example, the CICS program to be invoked, COMMAREA length, user ID, and password required:
   a. On the z/OS Explorer toolbar, select **Window** → **Preferences**.
   b. Select **CICS Transaction Gateway**.
   c. On the CICS Transaction Gateway preferences page, as shown in Figure 14-17 on page 346, follow these steps:
      i. Set the Program name to `EC01`.
      ii. Set the COMMAREA size (bytes) to `18`.
      iii. The User ID and Password can be left as blank, provided that the CICS server has not implemented link security.
      iv. Click **OK**.

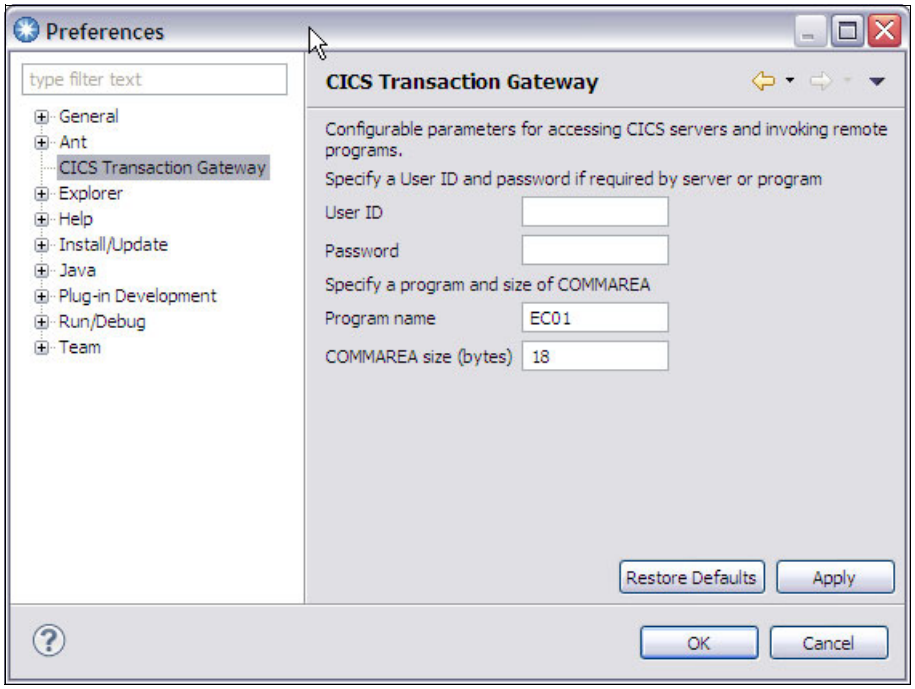


Figure 14-17 CICS TG preferences to customize the CICS program that is used to test the connection

2. From your preferred CICS TG Explorer view style, right-click the CICS server connection name, for example, `CICSTOR1` for Gateway daemon connection `zOS Test Gateway`, and click **Test CICS connection**, as shown in Figure 14-18.

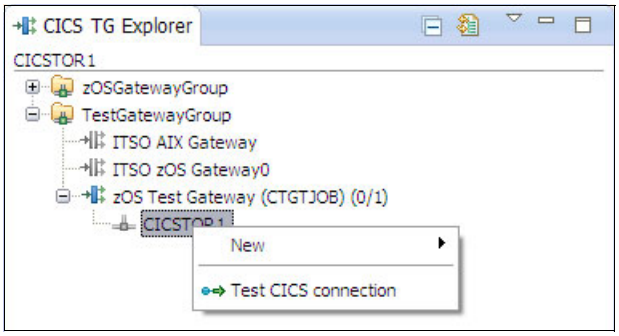


Figure 14-18 Test CICS connection dialog

3. The result is displayed in the Console view, as shown in Example 14-7.

Example 14-7 Example output from Test CICS connection action

```
Attempting to open connection to Gateway daemon using host
lp01.redbooks.ibm.com on port 2,009
Attempting to run program EC01 on CICS server CICSTOR1
Success: Time spent in ECI call was: 156 ms
```

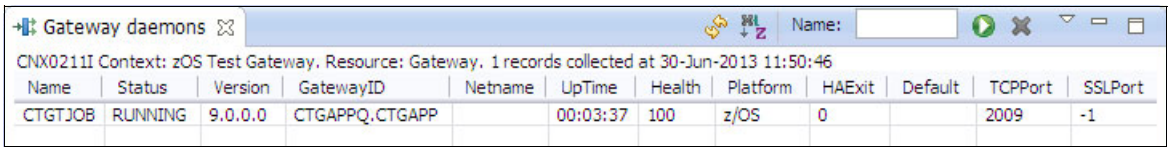
14.6.2 Gateway daemons view

The Gateway daemons view displays status and activity information for Gateway daemons. This view includes the values of properties specified in the configuration file and values obtained from the CICS TG startup, lifetime, and current statistics information. The Gateway daemons view displays information for the CICS TG connections that are currently selected in the CICS TG Explorer view. The CICS TG connection must be in the connected state.

To view the status and activity information for a Gateway daemon, follow these steps:

1. From the CICS TG Explorer view, ensure that the CICS TG connection, `zOS Test Gateway`, shows as connected (blue connection icon):
   - If the CICS TG connection is not currently connected (gray connection icon), right-click the connection name, for example, `zOS Test Gateway`, and select **Refresh status**.
   - If the connection fails, ensure that the Gateway daemon is running and listening on the statistics API port specified in the CICS TG connection definition.
2. From the CICS TG Explorer view, select the CICS TG connection, `zOS Test Gateway`. It is possible to select multiple Gateway daemon connections by pressing the Shift or Ctrl keys.
   - The Gateway daemons view now displays the status and activity information for the Gateway daemon as shown in Figure 14-19.

> **Note:** In this example, the value `-1` for SSLPort indicates that the SSL protocol handler was not defined for this Gateway daemon.



Name	Status	Version	GatewayID	Netname	UpTime	Health	Platform	HAExit	Default	TCPPort	SSLPort
CTGTJOB	RUNNING	9.0.0.0	CTGAPPQ.CTGAPP		00:03:37	100	z/OS	0		2009	-1

Figure 14-19 Gateway daemons view

The columns can be sorted by clicking the column headings, and the order can be rearranged by clicking and dragging the column headings. This view can be customized by using these options under the View Menu (triangle) icon:

- **Column Layout** → **Gateway** displays Gateway daemon property information.
- **Column Layout** → **Requests** displays information about requests made to that Gateway daemon.
- **Column Layout** → **Resources** displays information about the Gateway daemon resources.
- **Customize Columns** allows columns to be added or removed from the view.
- **Choose filter attributes** allows columns to be selected that can be used to filter requests. The default filter column displayed on the toolbar is for the Name. For Multiplatforms CICS TG connections, Name is the connection name. For z/OS CICS TG connections, Name is the Gateway daemon job or procedure name.

The **Reset Columns** option can be used to reset the view to the default options.

For a description of each column (property) value, see the CICS TG plug-in for CICS Explorer help. From the z/OS Explorer toolbar, select **Help** → **Help Contents** → **CICS TG Plug-in Users Guide** → **Views** → **Gateway daemons views**.

All of the status and activity information can also be viewed in the Properties view. This information is displayed when a single Gateway daemon is selected in the Gateway daemons view. The Properties view has two formats:

- Information is arranged in categories, for example, Client daemon, Gateway daemon, Connections, IPIC, and EXCI.
- Information is sorted in the alphabetical order of the properties.

To switch between the values, click the **Show categories** icon on the Properties view toolbar.

14.6.3 CICS connections view

The CICS connections view displays the status and activity information for the CICS server connections. The values are obtained from the CICS TG statistics information. The CICS connections view displays information for the CICS TG connections currently selected in the CICS TG Explorer view. The CICS TG must be connected. Multiple connections can be selected.

To view the status and activity information for the CICS server connections for a Gateway daemon, from the CICS TG Explorer view, select the CICS TG connection, for example, `zOS Test Gateway`. The CICS connections view now displays the status and activity information for the CICS server connections for the zOS Test Gateway.

Figure 14-20 shows the CICS server connection `CICSTOR1` status for the Gateway daemon `CTGTJOB`.

CICS connections Server:

CNX0211I Context: zOS Test Gateway, Resource: Connection, 1 records collected at 30-Jun-2013 11:49:41

Server	Gateway	Protocol	Default	AllocSess	FailSess	Applid	CommsFail	Requests	Response	Sent
CICSTOR1	CTGTJOB	IPIC		0	0	USIBMSC .S20351T1	0	6	4 ms	3.50 KB

Figure 14-20 CICS connections view

For a description of each of the column (property) values, see the CICS TG plug-in for CICS Explorer help. From the z/OS Explorer toolbar, select **Help** → **Help Contents** → **CICS TG Plug-in Users Guide** → **Views** → **CICS connections views**.

This concludes our brief tour of the CICS Transaction Gateway perspective that is provided with the IBM CICS Transaction Gateway plug-in for CICS Explorer. For more information about the options that are available with the views and statistics displayed, see the "*CICS TG Plug-in User Guide*" by clicking **Help** → **Help Contents**.

15

Troubleshooting techniques

This chapter provides techniques, materials, and utilities of use for troubleshooting configuration issues with IBM CICS Transaction Gateway (CICS TG).

This chapter contains the following topics:

15.1 Troubleshooting z/OS installations

This section includes various techniques, materials, and utilities specifically for troubleshooting with CICS TG for z/OS.

15.1.1 Obtaining CTGRRMS traced

The CICS TG for z/OS utility **ctgasi** includes the tracedump option `-td`, used to format the CTGRRMS internal trace table. This can be invoked directly from a z/OS UNIX shell such as OMVS. However, CICS TG for z/OS also includes sample JCL to invoke the z/OS UNIX utility in the `CTGASIS` member of the `SCTGSAMP` product data set. Example 15-1 shows a modified version CTGASIS, switching the `-s` (start) option for the `-td` option to dump the internal trace table.

Example 15-1 JCL to obtain a dump of the CTGRRMS internal trace table

```
//CTGASIS JOB (0),MSGCLASS=X,CLASS=A,NOTIFY=&SYSUID,REGION=96M,
// USER=CTGGW
/*JOBPARM SYSAFF=SC66
// SET CTGHLQ='CTG.V9ROMO'
// SET CTGHOME='/usr/lpp/cicstg/ctg900'
// SET LEOPTS='/'
//*
//* Main step: Stop the CTGRRMS services
//*
//CTG      EXEC PGM=CTGBATCH,
//         PARM='&LEOPTS.&CTGHOME./bin/ctgasi -td'
//STEPLIB  DD DSN=&CTGHLQ..SCTGLOAD,DISP=SHR
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDENV   DD DUMMY
//* Define CTGDBG DD DUMMY to enable CTGBATCH debug mode
//*CTGDBG   DD DUMMY
//*
//
```

If the TSO user ID used to submit the CTGASIS JCL does not have at least CONTROL or UPDATE access to the `CTG.RRMS.SERVICE` profile in the `FACILITY` class, the **ctgasi** command will fail with the error log messages shown in Example 15-2, and CTGBATCH completes with return code `8`.

Example 15-2 Output from ctgasi when the user ID has insufficient authority

```
CTG6209E ctgasi - user ID is not authorized, SAFRC = 00000008 00000008 00000000
CTG6202E ctgasi - CTGRRMS Services Address Space Initiator failed, RC=00000008
```

What else can I do with the ctgasi utility

Example 15-3 shows the help message output from the **ctgasi** utility.

Example 15-3 The help messages from the ctgasi utility

```
CTG6201I ctgasi - CTGRRMS Services Address Space Initiator
 CTG6200I (C) Copyright IBM Corporation 2005, 2010.  All rights reserved.
 CTG6221I Command format is:
```

```
CTG6222I    ctgasi [-r [-f] | -s [-f] | -td | -?] [-v]
CTG6223I Where the action performed  is modified by the optional parameters:
CTG6224I    -r  | -refresh   Stop the CTGRRMS services and start a new instance
CTG6225I    -s  | -shutdown  Stop the CTGRRMS services
CTG6226I    -td | -tracedump Dump a formatted initialization trace
CTG6227I    -f  | -force     Override registered user checks if required
CTG6228I    -v  | -verbose   Display extended messages while ctgasi is running
CTG6229I    -?               Display help on the command
CTG6230I ctgasi - completed successfully
```

You can use the refresh option to restart the CTGRRMS services, for example, if you apply CICS TG for z/OS maintenance to the `SCTGLINK(CTGINIT)` product load library.

15.2 Gateway daemon source of information

For a guide about the logs provided by the Gateway daemon, see 5.4.2, “Information available at run time” on page 79.

You can also add additional logging information as shown in the Table 15-1.

Table 15-1 Additional logging

Operating system	Additional logging available	How to code	location
Windows and UNIX and z/OS	Messages returned from CICS in IPIC error flows	`cicslogging=<on>` in the `GATEWAY` section of the configuration file	Gateway daemon error log
Windows and UNIX	Terminal installation and deletion requests	`terminstlogging=<Y>` in the `CLIENT` section of the configuration file	Client daemon log
Windows and UNIX and z/OS	Log each time that a client application connects to or disconnects from the Gateway daemon	`connectionlogging` in the `GATEWAY` section of the configuration file	Gateway daemon error log
z/OS	Messages are returned with Domain Name System (DNS) instead of the IP address	`dnsnames=<on>` in the `GATEWAY` section of the configuration file	Gateway daemon error log

Table 15-2 on page 352 provides a sample of the other log resources that are available.

Table 15-2 More available log resources

Platform	Resource	Location
Windows	Windows event log	`jvmdbg.log` in the *<product_data_path>*
UNIX and Linux	System error log	Run the **errp** command to create the system error log
AIX SNA	SNA error log	`/var/sna/sna.err`
Linux SNA	SNA error log	`/var/opt/ibm/sna/sna.err`
z/OS	MVS console log	
z/OS	CICS Transaction Server (CICS TS)	`MSGUSR`
z/OS	Logrec	`EREP`

15.3 Gateway Daemon and Java shared class caches

The Gateway daemon on z/OS allows you to have multiple Gateway daemons in a logical partition (LPAR) to consolidate the same class storage. In this section, we show several commands that can help you troubleshoot with CICS TG for z/OS.

15.3.1 Startup messages

The shared class cache is active, by default. Example 15-4 shows the message issued by the Gateway daemon when the class cache is successfully started.

Example 15-4 Output from Gateway daemon when the class cache is successfully started

```
CTG6134I The Gateway daemon JVM will use the class cache options
-Xshareclasses:name=cicstg900%g,groupAccess,nonpersistent,nonfatal
```

By default, the shared class cache is named based on the Gateway product and version. If you need to change it because you need to test two Java levels, use the following environment variable file with this coding:

```
CTGSTART_OPTS=-j-Xshareclasses:name=cicstgMM
```

Example 15-5 shows the message issued by the Gateway daemon when it is started.

Example 15-5 Output from Gateway daemon when the class cache name is overridden

```
CTG6134I The Gateway daemon JVM will use the class cache options
-Xshareclasses:name=cicstgMM
```

If you try to start the Gateway using an unsupported Java version, you receive the messages shown in Example 15-6 on page 353 and the Gateway daemon will not start.

Example 15-6 Output from Gateway daemon when an unsupported Java version is used

```
CTG6134I The Gateway daemon JVM will use the class cache options
-Xshareclasses:name=cicstg900%g,groupAccess,nonpersistent,nonfatal
CTG6109E The JVM is not at the required level to run this application
08/23/13 07:30:40:260 [0] CTG0827W CTGBATCH Child completed with a non-zero return
code 11
```

If you want to disable the shared class cache, you need to code this environment variable this way:

```
CTGSTART_OPTS=-j-Xshareclasses:none
```

Note: With Java V6, the option to code in `CTGSTART_OPTS` was `-noshareclasses`.

15.3.2 Useful commands to display the shared class cache

This section shows the use of a Java utility for troubleshooting with CICS TG for z/OS.

To check that the cache was created and the status of all the shared class cache is present in the LPAR, run the job shown in Example 15-7.

Example 15-7 JCL to display the status of the shared class cache in the system

```
//CTG5MM   JOB CLASS=A,USER=CTGGW,REGION=500M,NOTIFY=&SYSUID
/*JOBPARM SYSAFF=SC66
// SET CTGHOME='/pp/ctg900'
// SET ALLCACHE='-Xshareclasses:listAllCaches'
// SET JAVAHOME='/usr/lpp/java/J7.0_64/bin'
// SET CTGHLQ='CTG.V9R0M0'
// SET CTGUSR='CICSRS3.JCL(CTG5ENV)'
// SET LEOPTS='/'
//JAVCC   EXEC  PGM=CTGBATCH,
//       PARM='/&JAVAHOME./java &ALLCACHE.'
//STEPLIB  DD DSN=&CTGHLQ..SCTGLOAD,DISP=SHR
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDENV   DD DSN=&CTGUSR.,DISP=SHR
//
```

Example 15-8 on page 354 shows an example of the output.

Example 15-8 Output of listAllCaches

```
Listing all caches in cacheDir /tmp/javasharedresources/
Cache name       level         persistent  OS shmid       OS semid       last
detach time
Compatible shared caches
cicstgMM         Java7 64-bit  no          24581          12297          Fri Aug
23 07:23:14 2013
Incompatible shared caches
cicstg900CTGGRP  Java6 64-bit  no          24582          12298          Fri Aug
23 07:30:38 2013
cicstgMM         Java6 64-bit  no          24583          12299          Fri Aug
23 07:34:11 2013
ctgMM            Java6 64-bit  no          24584          12300          Fri Aug
23 07:35:25 2013
```

To check the class cache statistics, run the job shown in Example 15-7 on page 353.

Example 15-9 JCL to display the statistics of a specific shared class cache in the system

```
//CTG5MM   JOB CLASS=A,USER=CTGGW,REGION=500M,NOTIFY=&SYSUID
/*JOBPARM SYSAFF=SC66
// SET CTGHOME='/pp/ctg900'
// SET ALLCACHE='-Xshareclasses:name=cicstgMM,printStats'
// SET JAVAHOME='/usr/lpp/java/J7.0_64/bin'
// SET CTGHLQ='CTG.V9ROM0'
// SET CTGUSR='CICSRS3.JCL(CTG5ENV)'
// SET LEOPTS='/'
//JAVCC   EXEC  PGM=CTGBATCH,
//       PARM='/&JAVAHOME./java &ALLCACHE.'
//STEPLIB  DD DSN=&CTGHLQ..SCTGLOAD,DISP=SHR
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDENV   DD DSN=&CTGUSR.,DISP=SHR
//
```

Example 15-8 shows an example of the output.

Example 15-10 Output of printStats

```
Current statistics for cache "cicstgMM":
shared memory ID   = 24581
Cache created with:
 -Xnolinenumbers    = false
 BCI Enabled        = false
Cache contains only classes with line numbers
base address                        = 0x000002000011B700
end address                         = 0x0000020001100000
allocation pointer                  = 0x00000200006637D0
cache size                          = 16776656
free bytes                          = 7555388
ROMClass bytes                      = 5538000
AOT bytes                           = 1030452
Reserved space for AOT bytes        = -1
Maximum space for AOT bytes         = -1
JIT data bytes                      = 44388
```

```
Reserved space for JIT data bytes    = -1
Maximum space for JIT data bytes     = -1
Zip cache bytes                      = 1056048
Data bytes                           = 111824
Metadata bytes                       = 109356
Metadata % used                      = 1%
Class debug area size                = 1331200
Class debug area used bytes          = 514982
Class debug area % used              = 38%
Raw class data area size             = 0
Raw class data used bytes            = 0
Raw class data area % used           = 100%
# ROMClasses                         = 1789
# AOT Methods                        = 406
# Classpaths                         = 3
# URLs                               = 0
# Tokens                             = 0
# Zip caches                         = 22
# Stale classes                      = 0
% Stale classes                      = 0%
Cache is 51% full
```

To remove a specific class cache, run the job shown in Example 15-7 on page 353.

Example 15-11 JCL to remove a specific shared class cache from the system

```
//CTG5MM   JOB CLASS=A,USER=CTGGW,REGION=500M,NOTIFY=&SYSUID
/*JOBPARM SYSAFF=SC66
// SET CTGHOME='/pp/ctg900'
// SET ALLCACHE='-Xshareclasses:name=cicstgMM,destroy'
// SET JAVAHOME='/usr/lpp/java/J7.0_64/bin'
// SET CTGHLQ='CTG.V9ROM0'
// SET CTGUSR='CICSRS3.JCL(CTG5ENV)'
// SET LEOPTS='/'
//JAVCC   EXEC  PGM=CTGBATCH,
//       PARM='/&JAVAHOME./java &ALLCACHE.'
//STEPLIB  DD DSN=&CTGHLQ..SCTGLOAD,DISP=SHR
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDENV   DD DSN=&CTGUSR.,DISP=SHR
//
```

Example 15-8 on page 354 shows an example of the output.

Example 15-12 Output of the shared class cache destroy operation

```
JVMSHRC010I Shared cache "cicstgMM" is destroyed
```

Two files are created In the z/FS directory, `tmp/javasharedresources`, for each shared class cache as shown in Example 15-13 on page 356. The files are `semaphore` and `memory`.

Example 15-13 List of the files created in the Java temporary directory

```
$ [SC66] /SC66/tmp/javasharedresources: ls
C26OM2A64_memory_cicstg9OOCTGGRP_G21     C26OM2A64_semaphore_cicstgMM_G21
C26OM2A64_memory_cicstgMM_G21            C26OM2A64_semaphore_ctgMM_G21
C26OM2A64_memory_ctgMM_G21               C26OM3A32_memory_cicstg9OOCTGGRP_G21
C26OM2A64_semaphore_cicstg9OOCTGGRP_G21  C26OM3A32_semaphore_cicstg9OOCTGGRP_G21
```

It is possible, in certain circumstances, for residual information in the Java temporary directory to cause problems, and the content of this directory must be proactively deleted. It is then re-created on the first new reference to the class cache.

Note: All the Gateway daemon address spaces that use the same shared class cache must be shut down before you take actions on the shared class cache.

Part 4

Appendixes

This part includes additional information and examples.

A

Running the precompiled Java sample EciB1 from batch on z/OS

This appendix includes sample materials that allow a z/OS system administrator to use the precompiled Java sample program, `EciB1`. This program is useful for testing configurations, including Secure Sockets Layer (SSL) connections.

Use precompiled EciB1 from batch

This section provides a sample JCL JOB and z/OS UNIX shell script to run the precompiled CICS Transaction Gateway (CICS TG)-supplied sample program, `EciB1` (`com.ibm.samples.eci.EciB1`).

Normally, `EciB1` is run from a workstation and requires the user to provide keyboard input to select a target CICS server and, optionally, supply user ID/password security credentials. However, by using a combination of the **CTGBATCH** utility and a small z/OS UNIX shell script, `EciB1` can be driven from a batch JOB with the correct environment variable settings.

Note: The `EciB1` sample calls the CICS program `EC01` on a selected CICS server; therefore, `EC01` needs to be available on the target CICS server for a successful run. The COBOL source for `EC01` is in the CICS TG product installation path `samples/server/ec01.ccb`.

z/OS UNIX script

Example A-1 shows the z/OS UNIX shell script that we used to call the z/OS UNIX shell script, **ecib1**. On our test systems, this script file was stored on the z/OS UNIX file system in the `/u/cicsrs4/ctgtest/ecib1` directory. Read and execute permissions are set by using the z/OS UNIX command **chmod a+rx /u/cicsrs4/ctgtest/ecib1**. Permissions can also be set by using the TSO ISHELL menu system or by using the z/OS perspective in the CICS Explorer or z/OS Explorer.

Example A-1 z/OS UNIX shell script - ecib1

```
#!/bin/sh
#   ecib1 - run CICS TG Java sample com.ibm.ctg.samples.eci.EciB1
echo "ecib1: Executing EciB1 with parameters: " $ECIB1_OPTS
echo "ecib1: Classpath=" $CLASSPATH
echo "$ECIB1_SERVER_NUM
$ECIB1_USERID
$ECIB1_PASSWORD



" | java com.ibm.ctg.samples.eci.EciB1 $ECIB1_OPTS
```

Note: The three blank lines that follow "`$ECIB1_PASSWORD`" are intentional and required. They simulate keyboard input to the `EciB1` application, which otherwise stops to wait for user input.

This z/OS UNIX shell script is not intended to be used directly from the z/OS UNIX shell (OMVS) and relies on having the correct runtime environment provided. The required runtime environment is established by the partner batch JOB, `CTGECIB1`.

Batch JOB CTGECIB1

Example A-2 shows the batch JOB JCL that we used to call the z/OS UNIX shell script, **ecib1**. It uses the CTGBATCH utility program, and it provides the required environment variable values using the in-line STDENV DD statement.

Example A-2 Batch JOB JCL CTGECIB1

```
//CTGECIB1  JOB CLASS=E,MSGCLASS=A,MSGLEVEL=(2,0),REGION=128M
/*JOBPARM SYSAFF=SC66
//*********************************************************
//* CICS TG Gateway daemon Java compiler job              *
//*********************************************************
// SET CTGHLQ='CTG.V9ROMO'
// SET TARGET='/u/cicsrs4/ctgtest/ecib1'
//ECIB1  EXEC  PGM=CTGBATCH,
//       PARM='/&TARGET'
//STEPLIB  DD    DSN=&CTGHLQ..SCTGLOAD,DISP=SHR
//STDOUT   DD    SYSOUT=*
//STDERR   DD    SYSOUT=*
//STDENV   DD    *
PATH=/bin:/usr/bin:/usr/sbin:/usr/lpp/java/J7.0/bin
CLASSPATH=/usr/lpp/cicstg/ctg900/classes/ctgsamples.jar:\
/usr/lpp/cicstg/ctg900/classes/ctgclient.jar
ECIB1_OPTS=ssl://localhost 8070 \
/u/cicsrs4/ctgtest/eci_keyring.jks keyringpassw0rd
ECIB1_SERVER_NUM=1
ECIB1_USERID=CICSRS4
ECIB1_PASSWORD=CICSRS4
_BPX_SHAREAS=YES
/
```

The z/OS UNIX shell script **ctgecib1** relies on the values of four environment variables to control the program behavior:

- **ECIB1_OPTS**

  Command-line parameters for the EciB1 sample program.

- **ECIB1_SERVER_NUM**

  Simulated keyboard input to select the CICS server from the list of available CICS servers. The server is returned by the target Gateway daemon.

- **ECIB1_USERID**

  Optional value for the ECI request user ID.

- **ECIB1_PASSWORD**

  Optional value for the ECI request password (requires a user ID).

Note: The backslash ('\') continuation character is used in the STDENV data to specify values for environment variables that exceed the line length limits for JCL.

It also relies on the values of three more environment variables to define the required runtime resources and behavior:

- **PATH**

  Includes entries for the Java launcher and z/OS UNIX system commands

- **CLASSPATH**

  Requires the `ctgsamples.jar` and `ctgclient.jar` files

- **_BPX_SHAREAS**

  Not mandatory, but specifying `YES` forces created processes to run within a single address space

EciB1 command-line syntax

Specifying the correct value for the **ECIB1_OPTS** environment variable requires an understanding of the syntax expected by the `EciB1` sample program. Example A-3 shows the usage messages from `EciB1` that explain the meaning of the positional command-line parameters (separated by space characters).

Example A-3 Java sample EciB1 command-line parameters

```
CICS Transaction Gateway Basic ECI Sample 1
Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway URL]
                                           [Gateway port number]
                                           [SSL keyring]
                                           [SSL password]
To enable client tracing, run the sample with the following Java option:
  -Dgateway.T.trace=on
```

In Example A-2 on page 361, the following value is specified for ***ECIB1_OPTS***:

```
ssl://localhost 8070 /u/cicsrs4/ctgtest/eci_keyring.jks keyringpassw0rd
```

The sample program attempts to establish an SSL connection to a Gateway daemon that is running on the same LPAR and listening on port `8070` for the localhost adapter. The sample program uses the specified Java Key Store (JKS) file and password.

If the sample program is able to connect to the specified Gateway daemon, the simulated keyboard input is then used to select a target CICS server to flow the ECI request for CICS program `ECO1`.

B

Modifying the sample BasicCICSRequestExit

This appendix includes sample materials to allow a z/OS system administrator to modify the sample BasicCICSRequestExit (CICS Request Exit) that is provided by IBM CICS Transaction Gateway (CICS TG).

B.1 Introduction to the CICS request exit samples

CICS TG provides two request exit samples:

- **BasicCICSRequestExit**: This sample shows you how to implement a basic CICS request exit. It is based on a predefined server list. If the request arrives with a CICS server name present in the list, the remapping takes place. If the server name is not found, the request is not changed.
- **RoundRobinCICSRequestExit**: This sample shows you how to implement a basic CICS request exit to perform workload management. Each time that a request arrives to the Gateway, it returns the next CICS server from a predefined list. The request exit ignores the CICS server that is specified in the application request.

This chapter modifies the BasicCICSRequestExit Java sample.

The suggested development platform is the Eclipse development environment, where you can track your change management and where you have all the necessary tools to modify, compile, and test the Java programs. If Eclipse is not installed, but you still want to test the exit, you can use the Java compiler command **javac** directly to create the Java class.

B.2 Modify the sample CICS Request Exit

This section shows you how to modify and compile the BasicCICSRequestExit in z/OS by using commands in the UNIX System Services environment.

B.2.1 Modify the CICS request exit

To modify the sample, go in to the UNIX System Services environment using the TSO command **OMVS** or **ISHELL**.

The program source is in the product installation libraries:

`<install_path>/samples/java/com/ibm/ctg/samples/ha/BasicCICSRequestExit.java`

Copy the sample program into a directory to which you have write access. Edit the copy of the program using the **oedit** command to define an entry to map the CICS server value `CICSITSO` to server `CICSTOR1` by replacing the existing constructor code.

Example B-1 Constructor code

```
public BasicCICSRequestExit() {

   serverMappings = new Hashtable<String, String>();

serverMappings.put("CICSITSO", "CICSTOR1");

}
```

Note: You will receive a truncation warning: "`The data you are editing is variable length data with at least one record that ends with a blank`". Saving the data results in the removal of any trailing blanks from all records. You can issue the **PRESERVE ON** command if you do not want the blanks removed.

With the preceding changes in the BasicCICSRequestExit, all of the ECI requests that specify the server name `CICSITSO` will be routed to `CICSTOR1`.

B.3 Compile the Java sample request exit

This section shows two techniques to compile the modified exit.

B.3.1 Compile the Java sample request exit using UNIX System Services

The following steps show how to compile BasicCICSRequestExit in z/OS using UNIX System Services commands. When you complete all the changes to the sample, you must compile the source. Follow these steps:

1. Remember that the `JAVA_HOME` in your `.profile` needs to point to the correct Java level and that the Java `bin` directory must be included in the **`PATH`** environment variable. Example B-2 shows what you need to add.

   Example B-2 The .profile modification

   ```
   export JAVA_HOME=usr/lpp/java/J7.0
   export PATH=$JAVA_HOME/bin:$PATH
   ```

2. Ensure that the **`CLASSPATH`** environment variable contains the `ctgclient.jar`, for example, by using this command:

   export CLASSPATH=<*install_path*>/classes/ctgclient.jar

3. Compile the program by using the **`javac`** command:

   javac BasicCICSRequestExit.java

B.3.2 Compile the Java exit using the batch sample job

You can run the sample job that provided in this section to create the Java class. The intent of the provided sample is to show you how to create a Java class from a z/OS batch job simply. Follow these steps:

1. Give the correct permission to the Java source.

   chmod 644 BasicCICSRequestExit.java

   You can also use CICS Explorer, as shown in Figure B-1 on page 366.

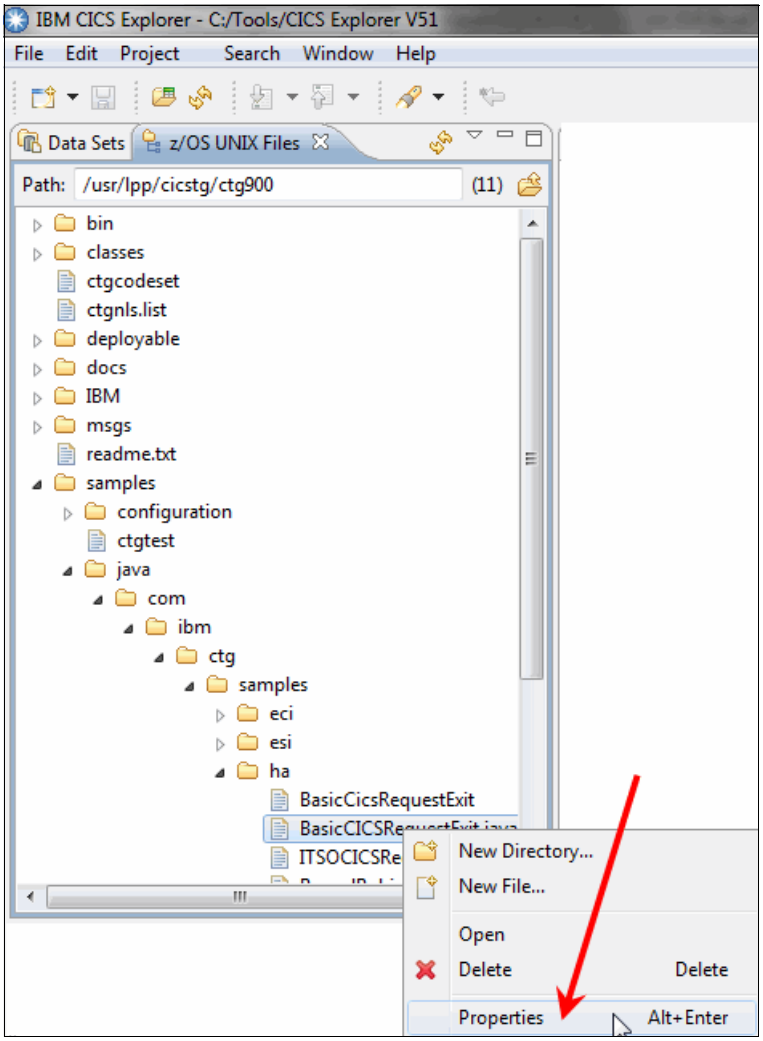


Figure B-1 Change the file property

You can decide the permission based on the minimum that is shown in Figure B-2 on page 367.

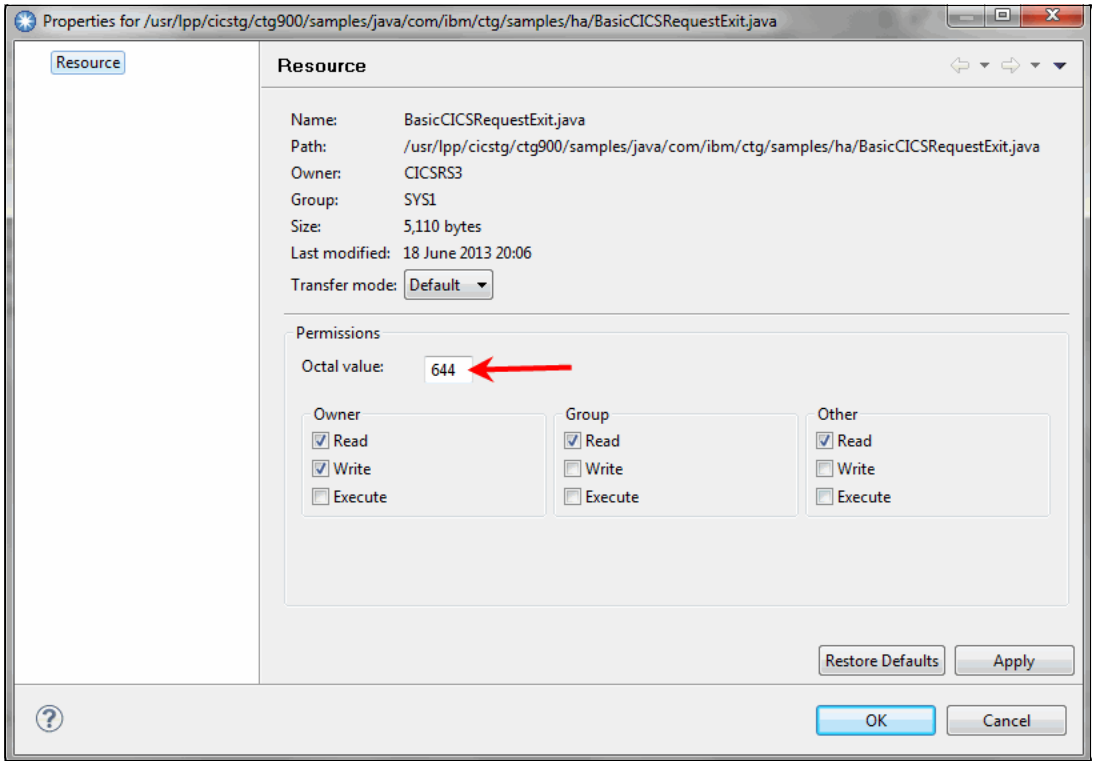


Figure B-2 Change the permission

2. Create the Example B-3 script in your user directory by using the **`oedit ctgjavac`** command.

Example B-3 The ctgjavac script

```
#!/bin/sh
echo '****************************************'
echo '*** CICS TG javac script ***'
echo '****************************************'
echo ''
echo CLASSPATH=${CLASSPATH}
echo SRCPATH=${SRCPATH}
echo SRCFILES=${SRCFILES}
echo OUTPATH=${OUTPATH}
echo command: ${JAVAHOME}/javac -verbose -d ${OUTPATH} -cp ${CLASSPATH}
${SRCPATH}/${SRCFILES} ${JAVAHOME}javac -d ${OUTPATH} -cp ${CLASSPATH}
${SRCPATH}/${SRCFILES}
```

You can use CICS Explorer to edit the file as showed in Figure B-3 on page 368.

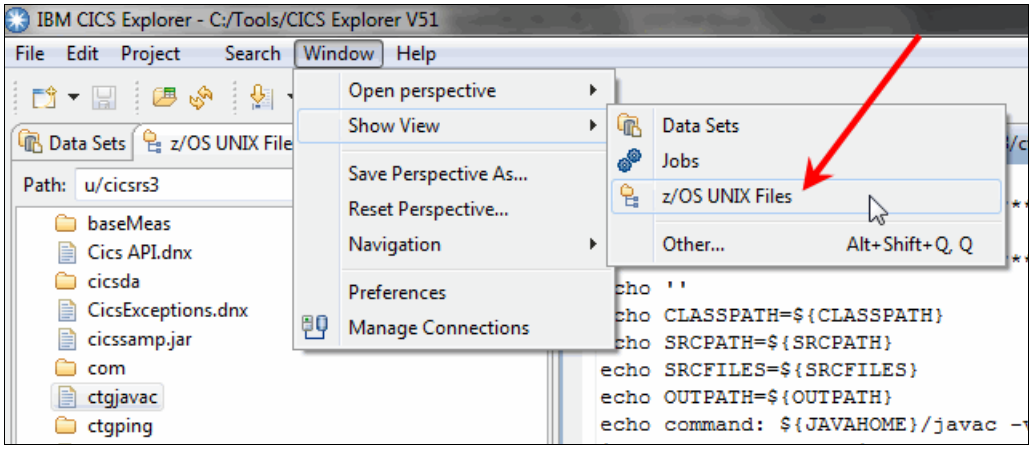


Figure B-3 Edit the z/OS UNIX files

3. Change the `javac` attributes:

   `chmod a+x /u/cicssr3/ctgjavac`

4. Create the batch job to start the script as shown in Example B-4.

   Example B-4 CTGBATCH sample

```
//CTGMM JOB CLASS=A,USER=CICSRS3,REGION=500M,NOTIFY=&SYSUID
//*********************************************************
//* CICS TG Gateway daemon Java compiler job *
//*********************************************************
//JAVAC EXEC PGM=BPXBATCH,
// PARM='SH /u/cicsrs3/ctgjavac'
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV DD *
PATH=/bin:/usr/bin:/usr/sbin:/java/J7.0/bin
JAVAHOME=/usr/lpp/java/J7.0/bin/
CLASSPATH=/usr/lpp/cicstg/ctg900/classes/ctgclient.jar:\
SRCPATH=/usr/lpp/cicstg/ctg900/samples/java/com/ibm/ctg/samples/ha/
SRCFILES=BasicCICSRequestExit.java
OUTPATH=/usr/lpp/cicstg/ctg900/userclass/
_BPX_SHAREAS=YES
/*
```

   The following explanations refer to Example B-4:

   – `SRCPATH` specifies the location of your Java source.

   – `SRCFILES` specifies the file name of your Java source.

   – `OUTPATH` specifies where you want to put the Java class that is created.

5. The CICS request exit must be activated in the Gateway daemon configuration file by setting this property:

   `CICSRequestExit = com.ibm.ctg.samples.ha.BasicCICSRequestExit`

6. When the Java class is ready, you need to make it available in the `CLASSPATH` by adding it in the Gateway environment variable.

B.4 SupportPac CA1T

SupportPac CA1T provides a good example of how to create high availability (HA) and request validation rules. Two workload balancing polices (*round-robin* policy or *failover* policy) are supplied, both of which use server rules from the HA configuration file to remap and retry failed requests. It is also possible to reload policy without restarting the Gateway.

SupportPac CA1T provides a combined workload balancing and request validation infrastructure to be used with CICS Transaction Gateway. It also allows validation rules to be applied on an inclusive basis or on an exclusive basis. When combined with a policy reload operation (the **CA1T_RELOAD** command), the validation rules can be used in exclusive mode to proactively reject requests for specific transactions, programs, or user IDs.

Configuration is based on a set of rules defined in the configuration files. Rules for workload balancing can be generic by each Gateway, based on server aliases, or on the payload type (channel or COMMAREA). Rules for request validation can be specified based on user IDs, transaction IDs, or program names. A configurable retry interval is provided to improve the efficiency of request distribution when you use workload balancing.

A configuration file failover mechanism is provided to enable failover between primary and secondary groups of CICS regions. All rules can be dynamically updated while the Gateway daemon is running. Workload balancing or request validation policies can be defined.

B.4.1 Workload balancing

Two workload balancing policies (round-robin or failover) are supplied. Both policies use server rules from the HA configuration file to remap and retry failed requests. Example B-5 shows a server rule that defines the server alias SERVER1 as mapping to three *actual* CICS servers CICSA, CICSB, and CICSC.

Example B-5 A CA1T server mapping rule

```
SERVER1=CICSA,CICSB,CICSC
```

The round-robin policy distributes requests evenly between CICS servers CICSA, CICSB, and CICSC. By contrast, the failover policy distributes requests in a hierarchical order so that all requests are sent to CICSA, provided that it is available. If CICSA is unavailable, requests are sent to CICSB. If CICSB is unavailable, requests are sent to CICSC.

Note: For either policy, after the exit determines a previous request failed with an error message that the request can be tried again, the failing CICS server is bypassed until the retry interval elapses.

If you use primary and secondary configuration files, the rules from the secondary file are activated whenever a request is to be retried and all CICS servers for the rule are unavailable. The rules from the primary configuration file are automatically reactivated after the duration set by the CTG_HAAUTOSWAP variable expires, or when a **CA1T_SWAP** command is issued.

B.4.2 Request validation

With CICS TG for z/OS V9.0, request validation is supported for both extended architecture (XA) and non-XA ECI requests by using the TRANSID, PROGRAM, and USERID rules to validate or reject each request. Before CICS TG for z/OS V9.0, only non-XA ECI requests achieved this form of request validation.

> **Note:** As of V9.0 of the CICS TG product suite, CICS TG for Multiplatforms and CICS TG Desktop Edition do not support XA requests in remote mode. This distinction is irrelevant.

Each request is validated against the rules defined in the HA configuration file in the Gateway daemon. Rejected requests are not tried again, and the following error is returned: `ECI_ERR_INVALID_CALL_TYPE (-14)`.

Example B-6 shows rules to reject requests by using the default `INCLUSIVE` validation mode.

Example B-6 CA1T request validation rules

```
TRANSID=CSMI
USERID=NULL
PROGRAM=PROG1,PROG2
```

This configuration rejects ECI requests that do not match all of the criteria:

- Use the default mirror transaction `CSMI`.
- Do not specify a user ID.
- Specify the CICS program `PROG1` or `PROG2`.

The readme file that is provided by SupportPac CA1T explains how to implement the CICS Request Exit, and it provides details about the full capabilities:

http://www.ibm.com/support/docview.wss?uid=swg24021484

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *CICS and SOA: Architecture and Integration Choices*, SG24-5466
- *CICS Transaction Gateway for z/OS Version 6.1*, SG24-7161
- *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133
- *Developing Connector Applications for CICS*, SG24-7714
- *Exploring Systems Monitoring for CICS Transaction Gateway V7.1 for z/OS*, SG24-7562
- *IBM CICS Performance Series: A Processor Usage Study of Ways into CICS*, REDP-4906
- *IBM z/OS V1R12 Communications Server TCP/IP Implementation: Volume 3 High Availability, Scalability, and Performance*, SG24-7898
- *IBM z/OS V1R13 Communications Server TCP/IP Implementation: Volume 3 High Availability, Scalability, and Performance*, SG24-7998
- *Securing Access to CICS Within an SOA*, SG24-5756
- *Threadsafe Considerations for CICS*, SG24-6351
- *Using CICS Transaction Gateway, High Availability, and the CICS Explorer,* REDP-4782
- *z/OS Identity Propagation*, SG24-7850
- *z/OS Version 1 Release 13 Implementation,* SG24-7946

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- Supported software for CICS Transaction Gateway (CICS TG) products:

  http://www.ibm.com/support/docview.wss?uid=swg21239203
- Shopz:

  http://www.ibm.com/software/shopzseries
- Scenarios:

  http://ibm.co/1tfLphd

- Summary of environment variables:
  http://ibm.co/1rxRuFr
- CICS Transaction Gateway family:
  http://www.ibm.com/software/products/us/en/cics-ctg/
- Passport Advantage and Passport Advantage Express:
  http://www.ibm.com/software/lotus/passportadvantage/
- Which API can be used? Multiplatform:
  http://ibm.co/1n2ltV6
- Which API can be used? Desktop:
  http://ibm.co/WnWuSN
- IBM CICS Performance Series: A Processor Usage Study of Ways into CICS:
  http://www.redbooks.ibm.com/abstracts/redp4906.html
- Identity propagation:
  http://ibm.co/1mYB2xG
- z/OS Identity Propagation:
  http://www.redbooks.ibm.com/abstracts/sg247850.html
- Implementing IPIC security:
  http://ibm.co/WjtHyu
- Autoinstalling IPIC connections; preliminary considerations:
  http://ibm.co/1rxNyVn
- Implementing MRO security:
  http://ibm.co/1n2j4cW
- Implementing LU6.2 security:
  http://ibm.co/1szvVqU
- Sample autoinstall user program to support predefined connection templates:
  http://ibm.co/1wG5Rbh
- CA76: CICS Transaction Gateway V9 SSL connectivity for .NET applications:
  http://www.ibm.com/support/docview.wss?uid=swg24034001
- Avoiding out-of-memory conditions:
  http://ibm.co/Wjxls7
- Hints and Tips for Java on z/OS:
  http://www-03.ibm.com/systems/z/os/zos/tools/java/faq/javafaq.html
- *CICS Transaction Server for z/OS V5.1 CICS Intercommunication Guide,* SC34-2858-01:
  http://ibm.co/1nGk8iR
- Syncpoint flows in CICS Transaction Server:
  http://ibm.co/1p37zCh
- CICS Transaction Gateway and Virtualization Environments:
  http://www.ibm.com/support/docview.wss?uid=swg21304599

- CICS Transaction Gateway and AIX workload partitions:
  http://ibm.co/1kEaHAo
- CA1T: High availability exits for use with CICS TG:
  http://www.ibm.com/support/docview.wss?uid=swg24021484
- ECI Client API exits:
  http://ibm.co/1qPwicK
- *z/OS V1R13.0 Security Server RACF Command Language Reference*, SA22-7687-16:
  http://www-01.ibm.com/support/docview.wss?uid=pub1sa22768716
- Configuring SSL security between a Java Client and the Gateway daemon (SC06):
  http://ibm.co/1p3aO7O
- ITU-T Recommendations:
  http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=X.509
- Configuring SSL security between a Java Client and the Gateway daemon (SC05):
  http://ibm.co/1zStHVw
- Using the CICS-supplied procedures to install application programs:
  http://ibm.co/UgHzID
- The conversion process:
  http://ibm.co/1szxnth
- Defining the conversion table:
  http://ibm.co/1uOTQRI
- Setting up profiles in RACF:
  http://ibm.co/1yEmB5s
- Building a key ring with certificates by using DFH$RING:
  http://ibm.co/1yEqJTg
- Using the CICS-supplied procedures to install application programs:
  http://ibm.co/UgHzID

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

The Complete Guide to CICS Transaction Gateway Volume 1 Configuration and Administration

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

The Complete Guide to CICS Transaction Gateway

Volume 1 Configuration and Administration

A practical guide to the capabilities and usage of the CICS TG product suite

Incremental setup from basic connectivity through to security and high availability

Focus on IPIC connectivity, high availability, WebSphere Application Server, and CICS Explorer

In this IBM Redbooks publication, you will gain an appreciation of the IBM CICS Transaction Gateway (CICS TG) product suite, based on key criteria, such as capabilities, scalability, platform, CICS server support, application language support, and licensing model.

Matching the requirements to available infrastructure and hardware choices requires an appreciation of the choices available. In this book, you will gain an understanding of those choices, and will be capable of choosing the appropriate CICS connection protocol, APIs for the applications, and security options. You will understand the services available to the application developer when using a chosen protocol.

You will then learn about how to implement CICS TG solutions, taking advantage of the latest capabilities, such as IPIC connectivity, high availability, and Dynamic Server Selection. Specific scenarios illustrate the usage of CICS TG for IBM z/OS, and CICS TG for Multiplatforms, with CICS Transaction Server for z/OS and IBM WebSphere Application Server, including connections in CICS, configuring simple end-to-end connectivity (all platforms) with verification for remote and local mode applications, and adding security, XA support, and high availability.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-8160-00 ISBN 0738439738