Redbooks

ibm.com/redbooks

The Next Generation of Distributed IBM CICS

Raghavendran Srinivasan

Janaki Sundar

Prashanth Bhat

Nageswararao V Gokavarapu

Ashwini Deshpande

IBM

Redbooks

International Technical Support Organization

The Next Generation of Distributed IBM CICS

June 2015

SG24-7185-01

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

Second Edition (June 2015)

This edition applies to Version 6.0, Release 6.0, of TXSeries for Multiplatforms (product number 5724-B44).

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 described in this document in other countries. Consult your local IBM representative for information about 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. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

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®
CICS®
DataPower®
DB2®
developerWorks®
HACMP™
IBM®
IMS™
Informix®
Lotus®
MQSeries®
RACF®
Rational®
Redbooks®
Redbooks (logo) ®
System i®
System p®
System z®
Tivoli®
TXSeries®
VisualAge®
VTAM®
WebSphere®
Worklight®
z/OS®

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.

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

This IBM® Redbooks® publication describes IBM TXSeries® for Multiplatforms, which is the premier IBM distributed transaction processing software for business-critical applications. Before describing distributed transaction processing in general, we introduce the most recent version of TXSeries for Multiplatforms. We focus on the following areas:

- The technical value of TXSeries for Multiplatforms
- New features in TXSeries for Multiplatforms
- Core components of TXSeries
- Common TXSeries deployment scenarios
- Deployment, development, and administrative choices
- Technical considerations

It also demonstrates enterprise integration with products, such as relational database management system (RDBMS), IBM WebSphere® MQ, and IBM WebSphere Application Server. In addition, it describes system customization, reviewing several features, such as capacity planning, backup and recovery, and high availability (HA).

We describe troubleshooting in TXSeries. We also provide details about migration from version to version for TXSeries. A migration checklist is included.

We demonstrate a sample application that we created, called *BigBlueBank*, its installation, and the server-side and client-side programs. Other topics in this book include application development and system administration considerations.

This book describes distributed IBM Customer Information Control System (IBM CICS®) solutions, and how best to develop distributed CICS applications.

Authors

This IBM Redbooks publication was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Bangalore Center.

Raghavendran Srinivasan is a TXSeries Level 3 support team lead and TXSeries test architect, and is based in India. He has 14 years of experience in information technology (IT) of which 11 years have been with the TXSeries for Multiplatforms product. His areas of expertise include distributed CICS systems.

Janaki Sundar is a Staff Software Engineer at IBM India Software Labs. She graduated with a B.Tech in Computer Science from the National Institute of Technology, Trichy. She has been with IBM and the TXSeries team for eight years. Her interests include distributed transaction concepts and middleware products. In her current role leading the test team, she provides technical reviews for the test plans, intended for functional and system verification testing of the product. She has also contributed to IBM developerWorks® articles and blogs, TXSeries product documentation, and a white paper on monitoring facilities in TXSeries.

Prashanth Bhat is a senior staff software engineer at TXSeries IBM Software Labs, India. He has been on the TXSeries team at IBM for the past 10 years. He has worked in different roles with TXSeries, including build, automation, installation, development, and web services enablement. His current focus is on enabling mobile and cloud workloads for TXSeries. He holds a Masters degree in Software Systems from the Birla Institute of Technology, Pilani, India.

Nageswararao V Gokavarapu is a software engineer. He has a Master of Engineering degree with Electronic Instrumentation as a specialization from the Electronics and Communication Engineering department from Andhra University, Vishakhapatnam, India. He has been with IBM India software labs for the last eight years. He has worked on distributed CICS transaction systems, modernizing applications with WebSphere eXtended Transaction Runtime (WXTR), and TXSeries for Multiplatforms software. He is a prolific contributor to the developerWorks community. His areas of interest include virtualization, transaction processing management, cloud computing, and UNIX systems.

Ashwini Deshpande has eight years of experience with IBM Software Group (SWG), Application and Integration Middleware Software. In TXSeries and WXTR development, Ashwini is working as a software developer. Her experience includes development and Level 3 support for TXSeries products. Her areas of interest include transaction processing management and UNIX systems.

Our special thanks to the following people for their contributions to this project:

- Anubhutil Kaushik, Software Designer: AIM Enterprise Transaction Systems
  IBM India
- Kasturi Mohan, Product Manager: TXSeries for Multiplatforms
  IBM India
- Gopalakrishnan P, Project Manager: TXSeries Development & Product Management
  IBM India
- Reshmi George, Technical Lead: TXSeries & WXTR Test in IBM Software Group, Application and Integration Middleware Software
  IBM India
- Ranjith E Raman, Release Manager: TXSeries (distributed CICS), WXTR
  IBM India

Our thanks to the following people for writing the first addition of this IBM Redbooks publication:

- Chris Rayns, Antonella Bertoletti, Iain Boyle, Yu Dong Kai, Nigel Eaton, Dan Litwin, Ken Ray, Hariharan Venkitachalam

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 client satisfaction, as you expand your network of technical contacts and relationships. Residencies run two - six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Learn more about the residency program, browse the residency index, and apply online:

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:

  ibm.com/redbooks

- Send your comments in an email:

  redbooks@us.ibm.com

- Mail your comments:

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

1

Introduction to IBM TXSeries for Multiplatforms

In this chapter, we introduce IBM TXSeries for Multiplatforms, a distributed online transaction processing (OLTP) product, its place in the client/server environment, and how it can be used. We also describe the objectives of this IBM Redbooks publication, and where this book aligns with other published material for TXSeries. This chapter also highlights the new features that are introduced in the most recent versions of TXSeries for Multiplatforms.

This chapter provides information about the following topics:

1.1 Overview of TXSeries for Multiplatforms

The chapter introduces TXSeries for Multiplatforms, the features and technical aspects of the current versions of TXSeries (V7.1 and V8.1), core components of TXSeries, and common deployment scenarios.

This section provides a comprehensive, high-level summary of TXSeries for Multiplatforms.

1.1.1 Basic view of TXSeries

TXSeries for Multiplatforms is a distributed processing solution for C, C++, Common Business Oriented Language (COBOL), Java, and PL/I in Cloud and traditional data centers. TXSeries provides a reliable, scalable, and highly available platform to develop, deploy, and host mission-critical applications. TXSeries for Multiplatforms integrates well into a mixed-language, multiplatform, service-oriented architecture (SOA) solution.

TXSeries has features that include two-phase commit, use of Transmission Control Protocol/Internet Protocol (TCP/IP) or Systems Network Architecture (SNA) communications, comprehensive security support, and many interfaces to other software products. These features enable client empowerment with minimal difficulty and seamless, transparent operations.

The foundation of TXSeries for Multiplatforms is IBM Customer Information Control System (IBM CICS), a product that operates on multiple platform types. CICS has built-in communication capability across SNA or Internet Protocol (IP) networks.

First developed for the mainframe environment, CICS has been extended to client platforms, such as Solaris, Microsoft Windows, IBM AIX®, Hewlett-Packard UNIX (HP-UX) on Intel Itanium, and Linux x86 operating systems. It runs with a comprehensive subset of the available application programming interface (API) and system programming interface (SPI) commands offered in IBM CICS Transaction Server (mainframe).

1.1.2 Practical uses

Among its many uses, TXSeries for Multiplatforms can be used to prolong the lifecycle of traditional systems by building a web-enabled front end. This enables communication of different systems as might be required, for instance, during a company merger, and relieves mainframe resource shortages by enabling the TXSeries to perform some of the processing.

This concept is explained in detail in the BigBlueBank demonstration application in Chapter 8, “The BigBlueBank sample application” on page 381.

1.1.3 Objective of this book

This book provides you a quick and easy-to-understand introduction to TXSeries for Multiplatforms and its capabilities and uses. This publication might not provide answers to *all* of your questions.

You can find more information in the TXSeries IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSAL2T/welcome

1.1.4 Achieving the objective

We intend to build and demonstrate TXSeries usage through instructions about how to establish a simple application network. We do this by using the BigBlueBank demonstration application across AIX, Windows, and IBM z/OS® platforms.

We also elucidate some of the theory with real-time scenarios behind TXSeries, In addition, we provide information about development and administration choices, troubleshooting, customization, and performance. We describe the development of distributed CICS applications, some of the supported languages, and distributed CICS solutions.

1.2 Introduction to distributed transaction processing

Every enterprise has a requirement for concurrent access to shared resources and data. Most often, these shared resources are distributed across geographical areas and even heterogeneous platforms. Applications written to perform operations on shared resources must maintain data integrity and consistency. To achieve this, it might be required that a group of operations on shared resources be treated as a single unit of work.

This grouping is also called a *transaction*, or a logical unit of work (LUW). a transaction is defined as a set of operations that transforms data from one consistent state to another. It has the following properties, popularly called ACID:

Atomicity A transaction’s changes are atomic. Either all operations that are part of the transaction happen, or none happen.

Consistency A transaction moves data between consistent states.

Isolation Even though transactions can be run concurrently, no transaction sees another transaction's work in progress. The transactions appear to run serially.

Durability After a transaction completes successfully, its changes survive subsequent failures.

Some examples of business transactions are banking, stock trading, a railway reservation system, and so on. Let us consider a railway reservation system. There are various steps involved to make a reservation:

1. Verify the seat availability.
2. Get payment clearance from the bank.
3. Update seat availability information.
4. Print and mail the ticket.

You can group these steps as a single transaction. The following issues are involved when building a transactional application:

- The application should ensure *atomicity* by reverting the whole operation if there is a failure in any one of the steps mentioned previously. For example, if there is a system failure while updating the seat availability information, the system rolls back step 2. In this case, the payment received must be credited back to the requester.
- The application and database servers should implement a mechanism whereby changes to databases are undone without loss of *consistency* of data. If a reservation failed for some reason, the user should see the correct information (bank balance and seat availability) during his next attempt.
- If there are multiple reservation requests for the same train, the system ensures *isolation* by letting only one transaction update the seat availability information at a time.
- The changes made by the application must be *durable*, meaning that it should survive subsequent failures. If the business logic and data are distributed, it must deal with network protocols and intersystem communications.

As you can see, it is complex to build a transactional application from scratch. A distributed transaction processing (DTP) system provides a framework for writing business applications. As part of the framework, the DTP system provides the following functions:

- System runtime functions. An execution environment that ensures the integrity, availability, and security of data. It also ensures fast response time and high transaction throughput.
- System administration functions. Administrative support that lets users configure, monitor, and manage their transaction systems.

- Application development functions. Transaction processing monitors provide functions for use in custom business applications, including functions to access data, to perform intersystem communications, and to design and manage the user interface (UI). By using all of the functionalities provided by a DTP system, an application developer can better concentrate on business logic without having to worry about other issues.

1.3 Key features of TXSeries for Multiplatforms

The most recent versions of TXSeries have significant new capabilities in the following areas:

- Platform coverage
- Operational efficiency
- Security
- Enterprise integration
- Serviceability integration and connectivity
- System resilience
- Application development tooling
- Problem determination tooling
- Web administration console
- Installation

Earlier versions of TXSeries made the following improvements:

- Vastly simplified infrastructure
- Enhanced the administration capabilities by introducing the web administration console
- Improved usability, system resilience, and interoperability
- Addressed many client requirements

1.3.1 Enhanced platform coverage

TXSeries for Multiplatforms has an enhanced platform coverage supporting Linux x86 architecture. TXSeries is supported on both 32-bit and 64-bit operating system versions of Linux. In a 64-bit operating system (OS), the product runs in 32-bit compatibility mode. However, you need to ensure that the programs are compiled using 32-bit flags and linked with 32-bit libraries.

1.3.2 Improved operational efficiency

TXSeries for Multiplatforms has included new features in version 8.1 that help in improving the efficiency, such as integrated monitoring through IBM Tivoli®, management of Workload Manager (WLM), and enabling non-root users to administer TXSeries. The section provides a brief description of these features.

Monitoring TXSeries using IBM Tivoli Monitoring

The TXSeries provides a centralized monitoring solution using IBM Tivoli Monitoring. You are able to monitor all of the TXSeries elements, such as regions, Structured File Server (SFS), Stand-Alone Remote Procedure Call daemon (SARPCD), and so on. You can also monitor all of the TXSeries deployments from a single Tivoli Enterprise Monitoring portal.

Working with CICS WLM

You can create a WLM configuration. The TXSeries Administration Console can be used to create and modify the WLM configuration file.

Enabling non-root users to perform TXSeries administrative tasks

TXSeries for Multiplatforms has simplified administration on UNIX and Linux platforms by enabling you, as a non-root user, to perform administrative tasks. As a non-root user, you can now stop, start, create, and destroy regions, Peer-to-Peer Communications (PPC) Gateway servers, and SFS. On Windows, you must be an administrator to perform CICS administration tasks.

Support for IPv6 dual stack

In TXSeries for Multiplatform V8.1, you can choose to communicate over IPv4, IPv6, or both (IPv4 and IPv6 in dual stack mode), depending on the availability in the operating system.

TRANCLASS feature for improved management of resources

The maximum number of transaction types is no longer restricted to 10. With TRANCLASS support, the restriction is now removed. You can also control the TXSeries tasks dispatches, based on priority, by grouping the transactions into transaction classes.

Ability to track transactions across TXSeries regions

The TXSeries for Multiplatforms supports a transaction tracking facility to help you track transactions spanning multiple regions based on TXSeries.

1.3.3 Enhanced security

TXSeries for Multiplatforms now supports passwords up to 100 characters in length. Passwords provided for connecting to the databases through extended architecture (XA) open strings are encrypted. This section provides a brief description of these features.

Support for passwords of greater than eight-character length

An eight-character limitation on the user definition password's length has been overcome with the `Password` and `NewPassword` options. These options can now accept passwords with a maximum of 100 characters in length.

Better handling of stored XA passwords to reduce security exposures

The TXSeries for Multiplatforms enables you to protect passwords with support for secure XA passwords. Secure XA passwords enable password protection, securing access to data in the resource manager. With the improved management of stored XA passwords, existing passwords are no longer saved as plain text, therefore providing you enhanced security.

1.3.4 Enhanced enterprise integration

Inbound web service requests are supported by TXSeries starting with V8.1. Through the web service facility, any TXSeries transaction can be started from any web services client. TXSeries also provides a trigger monitor facility through CICS MQ Adapter task initiator (CKTI) service transactions. The section provides a brief description of these features.

Improved integration with WebSphere MQ with support for CKTI and CKQC transactions

CKTI is the trigger monitor (or task initiator) supplied with TXSeries for Multiplatforms, and is used to start a transaction when the trigger conditions on any of its associated queues are met. CKTI starts a TXSeries transaction when an IBM WebSphere MQ trigger message is put to a specific queue, called *Initiation Queue*. You can use the **`CKQC`** commands to start and stop the CKTI manually. They can also be used to monitor CKTI queues.

Support for web services

Web services are self-contained, modular applications that can be described, published, located, and started over a network. Web services perform encapsulated business functions, ranging from simple request-reply to the entire business process interactions. The TXSeries Administration Console uses the WebSphere Application Server Liberty profile, a highly customizable, light-weight, and dynamic runtime environment.

The Liberty profile addresses the need for a minimal footprint, easy to configure, secure infrastructure for hosting web applications and web services. It also uses the Open Services Gateway Initiative (OSGi) Eclipse platform from IBM Lotus® Expeditor to provide a generic, platform-independent runtime environment. It functions as the web request gateway for the TXSeries web services.

1.3.5 Serviceability enhancements

TXSeries for Multiplatform V8.1 provides a facility to time out a transaction if its execution time exceeds a designated time, which is customizable. TXSeries by default warns the user, and also takes the stack of the process automatically if a transaction runs for a long time. This section briefly describes these features.

Transaction Timeout facility

TXSeries for Multiplatforms supports the Transaction Timeout facility. Previously, this facility was applicable for transactions in an intersystem communication scenario, and for conversational transactions only. You can now use this facility to set a timeout for transactions of any type.

Hang detection functionality

TXSeries for Multiplatforms provides a hang detection tool that detects potential hangs. It collects the required problem determination documents automatically.

1.3.6 Integration and connectivity

TXSeries V7.1 supports passing data of more than 32 kilobytes (KB) through channels and containers. This removes the restriction of allowing only 32 KB through the communication area (COMMAREA). Starting with TXSeries V7.1, the TXSeries can connect to mainframes through TCP/IP-based IP interconnectivity (IPIC) protocol for all distributed program link (DPL) operations across TXSeries and CICS Transaction Server (CICS TS).

IPIC supports Secure Sockets Layer (SSL) security for communication. The IPIC can also be used across two TXSeries regions. This section provides a brief description of these features.

Support for containers and channels

The limitation of transferring no more than 32 KB data between TXSeries and CICS TS programs using a COMMAREA has been overcome in versions later than V7.1. The channels and containers can now exchange data of unlimited size between these programs.

A *container* is a block of data designed for passing information between programs. Containers are grouped in sets called *channels*. It is a standard mechanism for exchanging data between programs. There is no limit on the number of containers that you can add to a channel, and the size of each container is limited only by the amount of storage available.

A channel can consist of multiple containers, so that it can be used to pass data in a more structured manner. Channels can be used by CICS application programs written in any of the languages supported by CICS. A server program can be written to handle multiple channels.

TXSeries also includes a group of new APIs that introduce containers and channels. Channels can be used by applications written in any of the programming languages supported by TXSeries. Options for data conversion are provided on the container and related API commands, providing a much simpler method than what was used with a COMMAREA.

In a COMMAREA, application data conversion is controlled by the system programmer. However, with the new enhancement, the applications are controlled by the application programmer.

1.3.7 Interoperability between CICS regions over TCP/IP

A new TCP/IP-based protocol support, IPIC, is introduced for communication across two CICS regions for DPL. Just as TXSeries CICS regions can communicate using this new protocol, TXSeries can also communicate with CICS TS regions. This provides an alternative to SNA in connecting to CICS TS. The existing IP-based support across TXSeries CICS regions continues to work the same as before.

1.3.8 Reliability, scalability, and availability

There are changes regarding memory isolation, several bug fixes, and WLM improvements in the product to improve reliability and availability aspects. This section provides a brief description of these features.

Isolation of memory usage

Memory allocated from a task-private pool is segregated from other heap allocations. This prevents applications from overwriting system memory across the boundary in most cases, and therefore improves the overall reliability and problem determination capability.

WLM enhancements for improved system flexibility and throughput

WLM has been enhanced to ensure higher availability of application-owning regions (AORs). This minimizes abnormal end of tasks (abends) in case of an AOR outage.

1.3.9 System Resiliency

TXSeries for Multiplatforms regions are resilient to failures in the environment on which it operates. TXSeries installer is also resilient to various failures, and the installer takes care of upgrading the version of TXSeries to the most recent level. The current installer provides a facility to repair any incorrect installations of TXSeries. This section provides a brief description of these features.

Recovery improvements to enhance system availability

CICS application manager has been enhanced to better handle recovery servers. Recovery servers now start more quickly, to enable faster recovery of tasks. The region is made more accessible.

Improved problem determination capacity

The problem determination capability has been enhanced with the introduction of CICS Application Probe Facility (task history logging), transaction life mapping, dump enhancements, and changes to the `cicsservice` utility.

Improved administration and usability

The Administration Console has been significantly improved to provide enhanced usability in selecting multiple programs for start and shutdown using a pop-up window. This helps improve performance and usability by moving the WLM attribute validations from the server to the client.

The Administration Console has intuitively reorganized the WLM view using a *groups* perspective. Also, the feature has improved granular control for users monitoring data, and an ability to monitor multiple regions concurrently. The feature is able to configure the CICS Application Probe facility.

Enhanced Installer

TXSeries Installer has been upgraded. The Installer is developed using Install Anywhere, which provides a seamless installation or uninstallation of TXSeries products and fix packs. The Installer takes care of necessary checks to prevent in-use files from getting overwritten. The installer provides detailed installation and uninstallation logs for verification and debugging purposes.

Enhancements to documentation

The TXSeries for Multiplatforms IBM Knowledge Center has been substantially enhanced. Earlier, it was in a book-oriented format, whereas now the information is presented as task-oriented user goals. This reflects the tasks that a user performs when using the product. Major revisions to the documentation library have also been made. Customers have a clear product overview, leading to faster and easier product deployment and administration.

The documentation library has been fully incorporated into the standard Eclipse-based IBM Knowledge Center, which provides many benefits. Such benefits include the ability to search all installed Eclipse-based IBM Knowledge Centers in a single search.

1.4 Technical value of TXSeries for Multiplatforms

TXSeries is a transaction server, and a rapid-deployment, transactional integration platform. It can provide the critical transaction and integration capabilities required to support your business goals.

As the IBM premier middleware product for DTP in traditional programming languages, TXSeries can serve as a critical component of your SOA. TXSeries enables you to connect otherwise disparate applications and data, helping you to run your business more smoothly, and serve your clients more effectively.

TXSeries delivers a managed environment for enterprise applications, enabling developers to focus on business logic rather than failure detection, failure recovery, and synchronizing access to shared data. It is the ideal program for both stand-alone deployments, and also larger solutions that require tight integration with other enterprise information systems. These enterprise systems can include CICS Transaction Server, IBM WebSphere Application Server, and centralized IBM System z® infrastructures.

It is the only distributed transaction-processing solution designed to enable you to scale your applications to CICS Transaction Server on the mainframe if your business requirements grow. In addition, because it follows the CICS programming paradigm, it is an ideal companion product for mainframe CICS users with distributed application or integration requirements.

As part of the CICS Family of products, TXSeries shares the value proposition of being robust, secure, extendable, and scalable. You can build new applications using procedural or object-oriented programming models across a range of supported programming languages. It also provides a set of tools to debug your applications, including source-level debugging.

TXSeries is designed to perform DTP. It runs on a wide range of distributed platforms, giving you the flexibility to choose the most suitable platform for your business. It runs on Linux x86, AIX, Solaris, HP-UX on Itanium, and Microsoft Windows. It supports full CICS intersystem communication with high-data integrity. You can distribute both function and data across platforms.

TXSeries enables transactional access to external resource managers for database and messaging systems. It supports several databases, such as IBM DB2®, Oracle, IBM Informix®, Microsoft SQL Server, and Sybase. Full-data integrity is provided through support of the XA interface, the industry-standard interface defined by X/Open for managing commitment and recovery of transactional data, including the two-phase commit process.

For more information about XA and X/Open standards, see the following website:

http://www.opengroup.org

TXSeries is positioned as both a rapid deployment *integration* server, and an entry-level *transaction* server, for systems other than Java Platform, Enterprise Edition (Java EE).

Interoperability with other CICS Family of products enables you to optimize your applications' cost, performance, and qualities of service (QoS). You can optimize these by choosing the appropriate set of CICS clients, gateways, and servers to match your business needs. You can port applications developed in TXSeries to CICS Transaction Server for z/OS, if your business requirements demand it.

To summarize, you can use TXSeries for Multiplatforms to complete the following tasks:

- Host business-critical, transactional CICS applications on distributed platforms in stand-alone deployments.
- Integrate data and applications in distributed solutions and enterprise systems, including CICS, IBM IMS™, DB2, and IBM WebSphere MQ.

- Run and extend CICS applications to the web by using CICS Transaction Gateway (CICS TG) and WebSphere Application Server.
- Extend CICS applications as web services using the TXSeries inbound web services capability.
- Reuse existing CICS applications and application programming skill sets in your organization consistent with corporate distributed platform policy.
- Java support, in the form of the Java class library for CICS (JCICS), assists interoperability with Java applications running outside of TXSeries. IBM CICS Foundation Classes (CFC) provide access to CICS facilities as C++ objects.
- Enhanced WLM provides simplified configuration and improved health monitoring.

1.5 Core components of TXSeries

TXSeries for Multiplatforms provides two core components:

- An entry-level CICS OLTP
- An integrated CICS SFS

1.5.1 Entry-level CICS online transaction processing

The entry-level CICS OLTP supports the base-level CICS API that provides the atomicity, consistency, isolation, and durability (ACID) properties. By delivering these, TXSeries enables developers to focus on the application logic, rather than failure detection, failure recovery, and synchronizing access to shared data. Without the ACID properties, integrity of an organization's data cannot be guaranteed.

1.5.2 Integrated CICS Structured File Server

The integrated CICS SFS is a Virtual Storage Access Method (VSAM) emulating record-oriented file system that can provide indexed, relative, and sequential access to file-based data. It enables developers to store fully recoverable, file-based data that can be processed in a batch environment.

The CICS SFS files can be shared among TXSeries, CICS TS, and applications other than CICS, such as IMS. This maximizes the ability to interoperate in an *enterprise environment*.

1.6 Common TXSeries deployment scenarios

TXSeries for Multiplatforms is ideally suited for deployment in two common business-value scenarios:

- As a rapid deployment, transactional integration server
- As a transaction server

1.6.1 TXSeries as a rapid deployment transactional integration server

TXSeries for Multiplatforms has extensive support for many enterprise information systems (EIS), including IBM CICS TS, IMS, DB2, WebSphere MQ, and WebSphere Application Server. It can use both TCP/IP and SNA-based communication protocols.

TXSeries has the ability to run intelligent business logic in a mid-tier environment, which supports the same languages and APIs as the systems that require the integration. This ability enables a complex integration solution to be deployed extremely rapidly. The following integration server deployment scenarios are the three most common:

- Consolidating mid-tier terminal server
- Intelligent mid-tier gateway between a Java EE application server
- Comprehensive mid-tier integration server, which is more than one EIS

A consolidating mid-tier terminal server

In this scenario, TXSeries acts as a relatively simple terminal concentrator between conventional terminal-based users and an EIS, such as CICS. TXSeries supports many locally connected, industry-standard graphical or screen-based terminals, web and mobile, providing users with fast responses and reduced network traffic.

Any transactions or requests for work that the users submit are routed to the EIS for execution. In general, all data and business logic remain on the EIS. The only exceptions are routing data required by TXSeries to make routing-based decisions, and local transactions involved in routing the requests to the EIS.

If we consider a banking solution where all of the applications run on mainframes, TXSeries can be used as a client concentrator in every branch office of the bank, as shown in Figure 1-1 on page 15. Bank staff or customer service representatives (CSRs) in the branch office use industry-specific graphical or screen-based terminals connected to the local TXSeries server.

Any transactions that the users enter are routed to the CICS Transaction Server in the bank's central data center for execution. TXSeries effectively takes the entire client load, and reduces the number of connections to mainframe.

Key features of this deployment

The following list describes the key features of this deployment, shown in Figure 1-1:

- Reduces the number of connections to the EIS, because many terminal connections are replaced with one connection from TXSeries
- Protects EIS from client-originated network disruptions
- Takes advantage of the wide range of terminals supported by TXSeries
- Prevents duplication of business data on the TXSeries environment

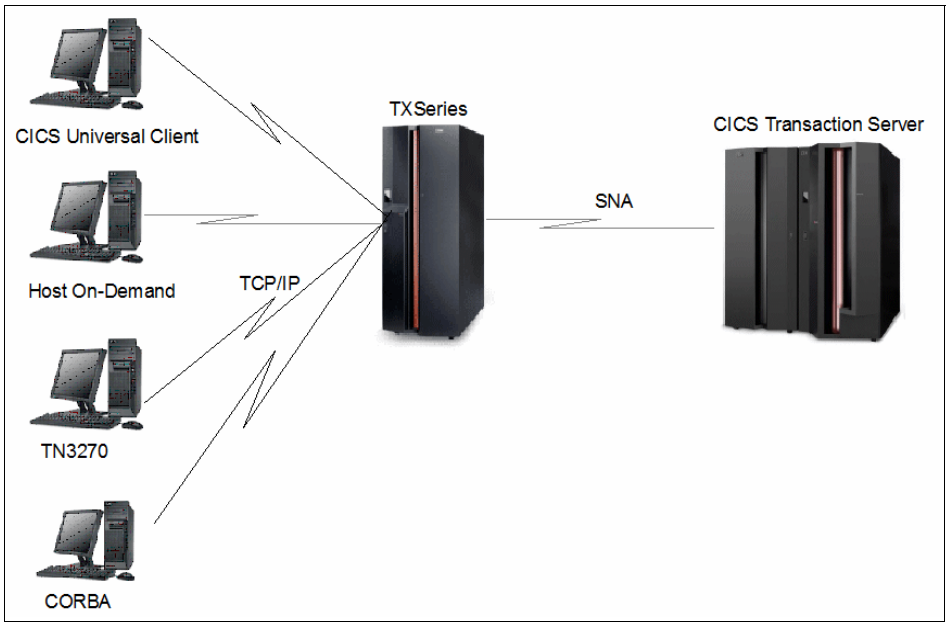


Figure 1-1 Consolidating mid-tier terminal server

An intelligent mid-tier gateway

TXSeries can provide excellent integration with one or more EIS. Its ability to handle transactions makes it an ideal intelligent mid-tier gateway. There is a vast range of business application intelligence that you can build into TXSeries:

- Perform security checks.
- Complete intelligent routing based on client request.
- Run decision-making transactions to achieve intelligent routing.

The conventional terminal-based client interfaces can be extended to the Internet using the facilities of WebSphere Application Server and CICS Transaction Gateway. This scenario is widely used in very large deployments, where several mission-critical applications run on mainframes.

This scenario builds upon the previous scenario in three important areas:

- Using the facilities offered by WebSphere Application Server and CICS TG, organizations can extend the conventional terminal-based interfaces provided by TXSeries to include intuitive and rich web interfaces.
- Acting as an entry-level online transaction processing system, TXSeries is capable of making intelligent business decisions on its own.
- TXSeries can access and update multiple EIS in a transactional, coordinated manner, without losing or corrupting data.

By combining these three capabilities with the previous scenario (see Figure 1-2 on page 17), TXSeries can act as an intelligent gateway between many different users and multiple EIS. TXSeries can now make business decisions based on business data retrieved from multiple EIS, and present the decisions to users in several different ways.

Consider an insurance company where most of the critical data is stored on VSAM and IMS. All of the vital business logic runs on CICS Transaction Server, as shown in Figure 1-2 on page 17. This company can provide a rich web interface for customers to transact business, such as getting insurance quotes or paying premiums online. It can use WebSphere to manage the presentation logic, giving the users a rich web interface.

Agents can use conventional terminals connected directly to TXSeries, and get access to a greater range of business transactions, such as listing their client portfolio. TXSeries plugs into the middle tier, and performs intelligent routing to either IMS or CICS TS. For intelligent routing, TXSeries completes the following functions before routing the request:

- Authenticates clients
- Caches data locally on a DB2 server to perform data validation
- Routes to IMS or CICS TS based on client request

Key features of this deployment

The following list describes the key features of this deployment:

- Reduced resource consumption on the EIS, because TXSeries takes over several business operations
- Intuitive and rich web-based interfaces available to users
- Consolidated data from multiple EIS systems before presentation to users

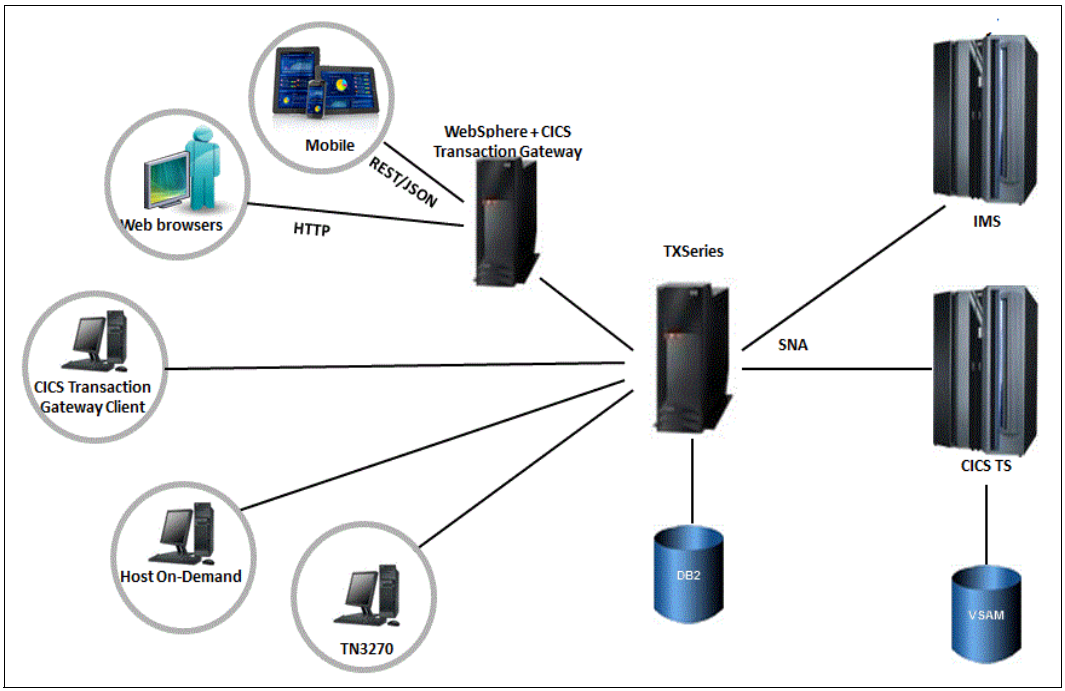


Figure 1-2 An intelligent mid-tier gateway

A comprehensive mid-tier integration server

As organizations change and evolve, either organically or through mergers and acquisitions, the location of critical business data can become dispersed on several heterogeneous EIS systems. TXSeries is an effective mid-tier integration solution for consolidating, summarizing, and presenting essential business information to parts of the organization.

Information can be seamlessly summarized in a single location. This integration avoids the need to consolidate existing EIS systems or duplicate essential business data across multiple systems.

Consider a large bank, Bank A, which decides to acquire two other banks, Bank B and Bank C. Bank A has more than 3000 branches across the country. Most of the critical business logic runs on CICS TS. It also provides a rich web interface for its customers. Bank B also runs its core business on CICS TS, but supports only 3270 client-based terminals.

Bank C has a total non-IBM solution. Because of the acquisition, there is a requirement to provide senior management with consolidated data and information taken from all three existing heterogeneous EIS.

The extensive integration capabilities of TXSeries make it an ideal choice as a mid-tier integration server. As shown in Figure 1-3, TXSeries integrates with CICS Transaction Server on z/OS (Bank A and Bank B), IBM MQSeries®, and a non-IBM solution (Bank C) to fulfill all of the requirements.

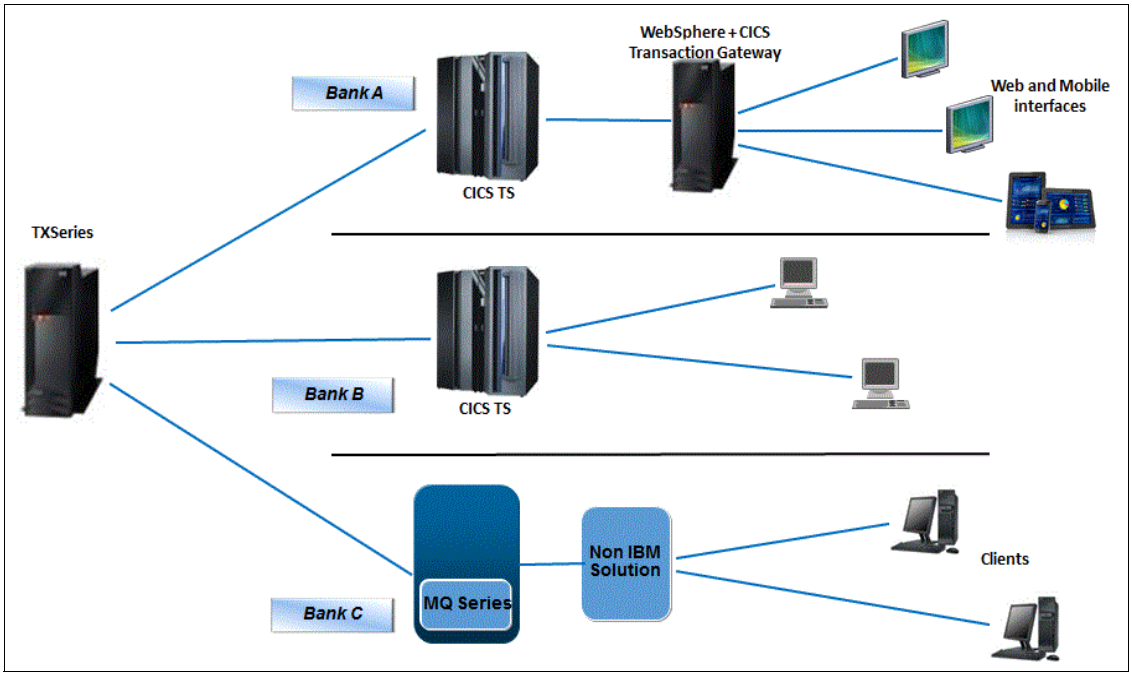


Figure 1-3 A comprehensive mid-tier integration server

Key features of this deployment

The following list describes the key features of this deployment:

- Essential business information taken from multiple EIS is consolidated, summarized, and made available in one place.
- Rapid integration of heterogeneous EIS systems occurs, without the need to consolidate these systems.

1.6.2 TXSeries as a transaction server

TXSeries for Multiplatforms provides base level CICS programming interfaces, enabling industry-specific COBOL, C, C++, PL/I, and Java specialists to create simple solutions for transaction processing. It is the only distributed OLTP solution that IBM has designed to enable you to scale up to CICS TS on the mainframe if your business requirements grow. TXSeries supports screen-based interfaces and graphical user interfaces, depending on the business requirement.

You can access data from the integrated CICS SFS, from a local or a remote relational database management system (RDBMS), such as DB2, or from a messaging product, such as IBM MQ. With excellent enterprise integration support, TXSeries is ideal for creating mainframe added value or stand-alone DTP solutions.

The following three application server deployment scenarios are the most common:

- An entry-level, stand-alone transaction server
- Server other than Java EE in a mixed-workload environment
- A distributed server with mainframe connectivity

An entry-level, stand-alone transaction server

In this scenario, TXSeries acts as an entry-level online transaction processing system running applications written in Cobol, C, C++, PL/I, and Java, as shown in Figure 1-4 on page 20. All application data is stored locally in one of the supported RDBMS or the CICS SFS. Any CICS internal data required for operational integrity of the CICS system is stored in the SFS, DB2, or Oracle.

TXSeries has full responsibility for applications running on the system, and is not dependent on any other EIS for data or transactional work. It provides a wide range of functions, tools, and facilities to help users and system administrators run their applications and manage the environment.

This scenario has access to all of the features of TXSeries, including transactional capabilities, authentication and authorization security, and data access. It is ideally suited for small-scale deployments, such as a small finance company, automobile dealers, and container shipping companies.

As an example, consider a business solution for a small automobile dealer operating several franchises in a city. Assume that the dealer has one central office and several car showrooms. Some of the needs that TXSeries can satisfy might include the following requirements:

- Most users of the system are in a limited number of locations and use either web applications Mobile applications or 3270-based terminals.
- Any client orders are tracked and managed from the time the order is placed to the delivery of the car. An inventory of spare parts stored in the various workshops with the ability to reorder parts from the car manufacturers.
- The automotive dealer either writes the applications or, more likely, purchases a custom-built application based on TXSeries and DB2.

Key features of this deployment

The following list describes the key features of this deployment, as shown in Figure 1-4:

- Cost-effective and robust solution for deploying and running business transactions
- Ideal solution for clients who do not have CICS or large-scale EIS systems, but require the transaction handling qualities of service offered by CICS

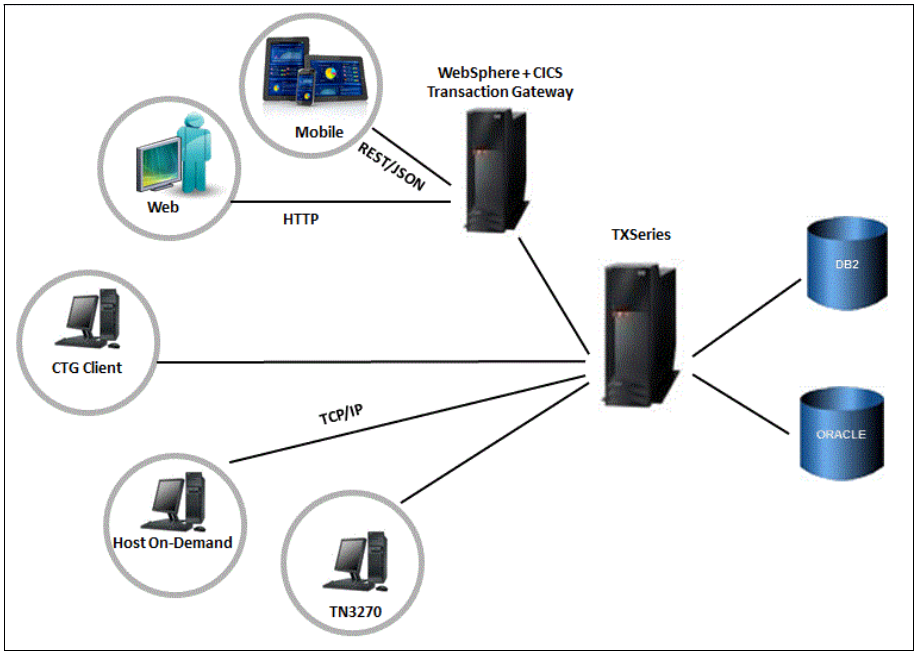


Figure 1-4 Entry-level, stand-alone transaction server

A server other than Java EE in a mixed workload environment

TXSeries supports a wide range of industry-specific graphical or screen-based terminal interfaces, web and mobile, enabling users to make the most of its exceptional transaction handling properties. Integrating TXSeries with WebSphere Application Server using the CICS Transaction Gateway enables composite applications with sophisticated, feature-rich UIs to become the front end of TXSeries applications.

The CICS TG even connects TXSeries to the most recent WebSphere-based SOA products, such as WebSphere Enterprise Service Bus, enabling TXSeries applications to become endpoints in an SOA.

This SOA functionality enables the TXSeries infrastructure to be extended to access new markets and new types of users, as shown in Figure 1-5. Continuing our previous example of an automobile dealer, the company can provide a web interface for its customers to book a slot for servicing their car, view bills online, or order spare parts.

With minimum changes to the existing applications, the dealer can extend to the web by writing some presentation logic running on WebSphere Application Server and connect to TXSeries using the CICS TG. All existing employees and users can still continue to access the system using terminal interfaces.

Key features of this deployment

The following list describes the key features of this deployment:

- You can access TXSeries applications from a Java-based environment.
- Transactional integrity is propagated automatically by CICS TG from WebSphere Application Server to TXSeries. This ensures that web-based users receive the same, or better, QoS from TXSeries as conventional terminal-based users.
- Business applications are written using a combination of Java EE and traditional CICS programming languages.

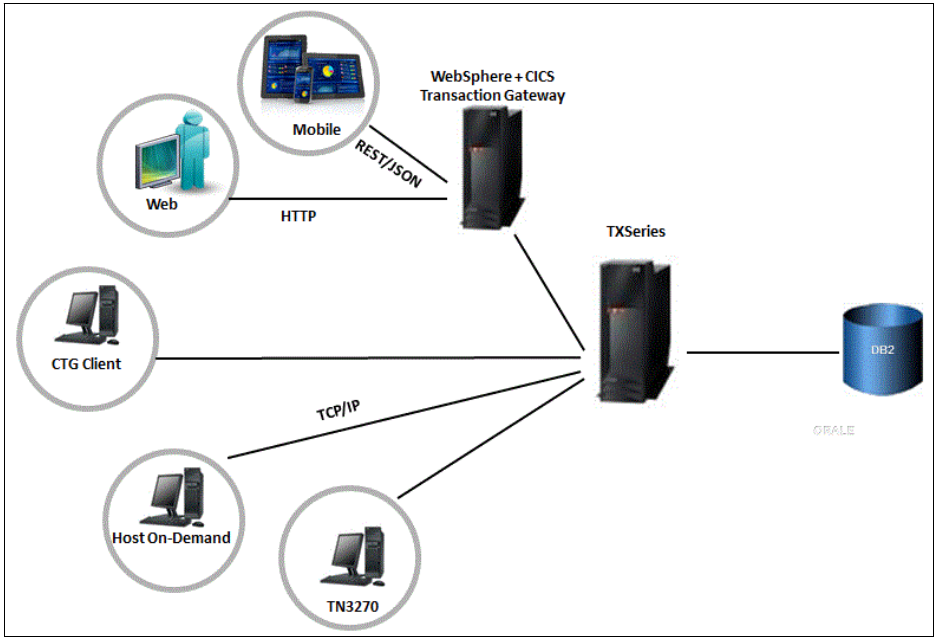


Figure 1-5 Non-Java EE server in a mixed workload environment

A distributed server with mainframe connectivity

This scenario covers almost the complete spectrum, because it uses the integration of TXSeries with WebSphere Application Server, other members of the CICS Family, and other EIS systems.

TXSeries can be used as a full transactional system running applications on behalf of conventional users and Internet-based users with WebSphere Application Server. In either an online or batch capacity, TXSeries can communicate with other CICS systems or EIS to synchronize data or start mission-critical functions only available on an EIS, as shown in Figure 1-6.

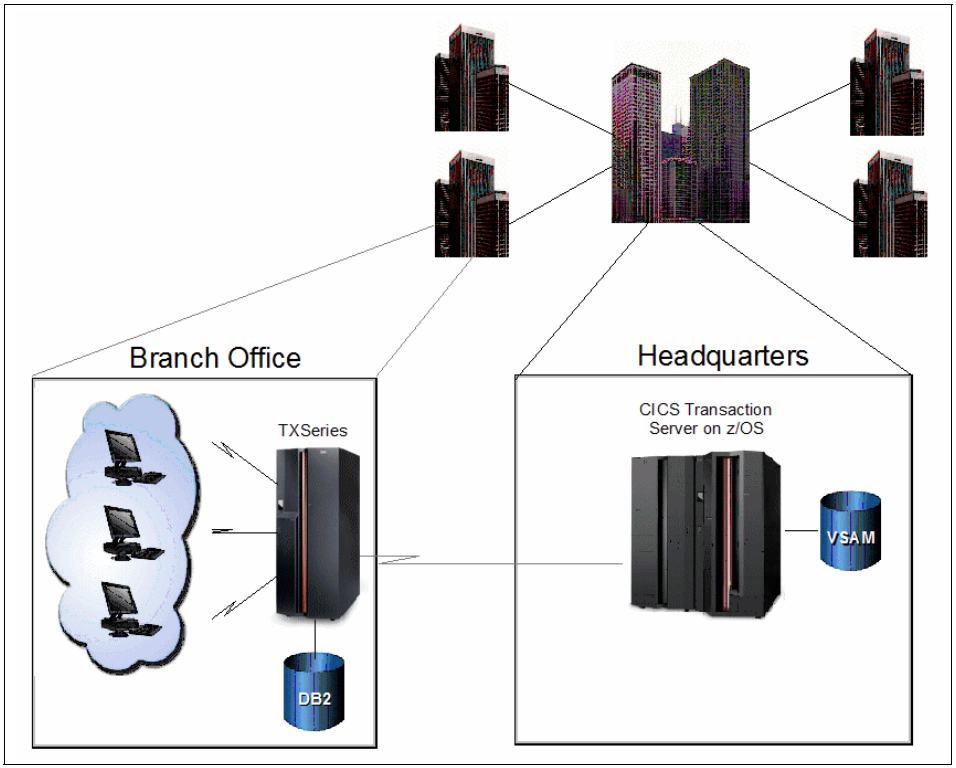


Figure 1-6 TXSeries can communicate with CICS and EIS systems to synchronize data or start mission-critical functions

An example for this deployment is a large retail organization that sells various goods through outlets and shops, while also providing Internet-based shopping. TXSeries can act as a server in each shop or outlet. In this scenario, TXSeries manages all local transactions, such as order processing, inventory management, and shop-specific data.

All shop sales representatives use terminal-based interfaces to interact with the system components:

- TXSeries
- Web browsers
- Mobile
- Hypertext Transfer Protocol (HTTP)
- DB2
- WebSphere and CICS
- CICS Transaction Gateway
- CICS Universal Client
- IBM Rational® Host On-Demand
- TN3270

Each of these shops or outlets is connected to a single mainframe (Figure 1-7 on page 24) in the headquarters. The EIS manages the entire business, including the business tasks, such as corporate-level data, corporate report generation, invoicing, and salary processing. Any retail-related data, such as product pricing and promotional information, is synchronized between the shops and the EIS by overnight batch updates.

Internet-based users connect through a web-based interface directly to the main EIS. However, information is distributed to local shops, enabling customers the option of collecting their orders from a shop rather than waiting for delivery.

Figure 1-7 shows the distributed server with mainframe connectivity solution.

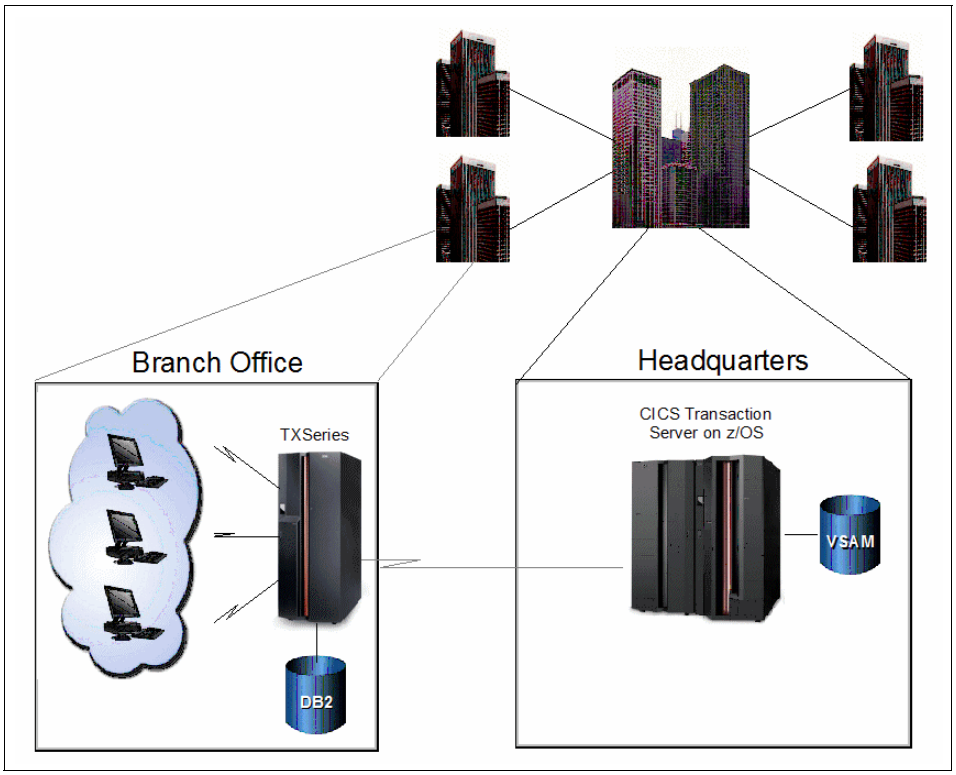


Figure 1-7 Distributed server with mainframe connectivity

Key features of this deployment

The following list describes the key features of this deployment:

- There is full end-to-end integration between WebSphere Application Server, CICS Transaction Gateway, CICS, other EIS, and TXSeries.
- Solutions can be simple or complex, depending upon the needs of the organization and the demands of the users.
- As the organization grows or changes, the architecture can remain constant while using the flexibility of the CICS Family to select the most appropriate combination of products for the users' needs.

1.7 Deployment, development, and administration choices

TXSeries, being a true distributed transaction processor, enables both data and functions to be distributed across heterogeneous systems. This enables you to use TXSeries in several different architectures. To facilitate this flexibility, TXSeries provides architects, developers, and administrators with several choices. This section describes some of the choices available to TXSeries clients in the areas of deployment, development, and administration.

1.7.1 Deployment choices

This section describes some of the choices and options available to architects of a TXSeries-based solution.

Choosing a Virtual Storage Access Method file system

For VSAM data, transient data, and temporary storage queues, TXSeries can either use the integrated CICS SFS provided with it, or it can use the RDBMS.

Currently, DB2 and Oracle are supported databases for VSAM data. TXSeries does support a wider range of RDBMS for conventional database data. When deployed, it is possible to change a TXSeries system from using the CICS SFS for VSAM data to RDBMS, and vice versa. However, do not attempt this regularly. The best option is to make a decision and implement it.

The following sections briefly summarize the advantages and disadvantages of using the CICS SFS for VSAM data, compared to RDBMS.

Advantages of SFS

The following items are advantages of using SFS:

- Included with TXSeries and does not require any additional licenses
- Easy to create the initial configuration and deploy VSAM files
- Supports all file options (key-sequenced data set (KSDS), entry-sequenced data set (ESDS), relative record data set (RRDS)), temporary storage (TS) queues, and transient data (TD) queues
- Supports both recoverable and unrecoverable files

Disadvantages of SFS

The following items are advantages of using SFS:

- Requires additional skills to manage and tune
- Limited options to back up and restore data to media outside of SFS control

Advantages of RDBMS

The following items are advantages of using RDBMS:

- Because it is widely used, skills are easily available.
- It is easy to back up and restore data.

Disadvantages of RDBMS

The following items are disadvantages of using RDBMS:

- DB2 or Oracle licenses must be purchased separately apart from TXSeries.
- The concept of unrecoverable files does not exist.

Both SFS and RDBMS can store large volumes of data, and both enable batch programs to access the data. Both options have their advantages and disadvantages. Your choice probably depends on several factors, such as the availability of licenses and the necessary skills within your organization.

Location of file system

TXSeries SFS must physically be on the same machine as the TXSeries server. This leaves several options for physically locating CICS regions and RDBMS servers. Depending on the capacity of the server and the skill set available in the organization, RDBMS can either run on the same machine as TXSeries or remotely.

Choosing the type of security

TXSeries provides the flexibility of choosing between CICS integrated security features or external modules for authentication and authorization. With TXSeries integrated security, each CICS region authenticates users and incoming communication, and authorizes access to the resources of that system. All of the authentication and authorization depends on the user definition in the runtime database.

You can enhance or replace authorization services by using an external security manager (ESM) that is called from CICS. Similarly, you can enhance or replace authentication services by using an external authentication manager (EAM) that is called from CICS.

With EAM and ESM, the possibilities for security management and integration are greatly increased. An EAM module is included that uses Lightweight Directory Access Protocol (LDAP) to integrate with Resource Access Control Facility (IBM RACF®). With RACF integration, all the user authentication and authorization can be centralized on an EIS.

Your choice of integrated security or external modules depends on several factors, including the need to centralize user authentication and authorization.

Choosing the network protocol for intersystem communication

TXSeries supports intersystem communication (ISC) between a local TXSeries region and the following system components:

- Other TXSeries regions
- IBM CICS Transaction Server on z/OS

TXSeries can communicate with remote systems using either TCP/IP or SNA protocol. ISC over TCP/IP or SNA can be used between two TXSeries regions, or across CICS TS regions. TXSeries supports communication requiring Synchronization level 1 and Synchronization level 2.

Synchronization level 2 across an SNA connection requires a separately purchased communications product, such as IBM Communications Server, to be installed on the same machine as TXSeries. Your final choice of protocol for ISC depends on several factors, such as the need to access remote systems, existing networks, and required synchronization levels.

Choosing an operating system for deployment

TXSeries is supported on UNIX style systems (AIX, Solaris, Linux x86, and HP-UX on Itanium) and on Microsoft Windows. As with all production systems, the final choice about which OS to run TXSeries on depends on several factors:

- The number of physical machines that are required for functions, such as production usage, testing, backup, and development
- The number of physical locations, such as separate data centers for locating machines
- The ability to consolidate separate machines into a smaller number of large machines
- Operating system features, such as security and backup, required by the administrators and users
- Availability of different software required by the organization
- Number and type of systems already deployed within the organization

Due to the prevalence of existing machines, the choice of an OS can be straightforward for some organizations. For others, it can be more difficult.

1.7.2 Development choices

When designing and developing a CICS application based on TXSeries, developers have many choices about how a CICS application behaves. In many cases, poor performance and behavior in CICS can typically be traced back to an inappropriate design decision at development time. This section offers some of the choices and options available to developers of a solution based on TXSeries.

In general, the choices offered in this section are not exclusive to TXSeries. They apply to all applications based on CICS, regardless of the CICS product at which the application is targeted.

Separation of logic

An important principle of business application design is to separate the application logic into *components*. For example, broad levels of separation can be presentation logic, business logic, data handling, communications, error handling, and so on. With components, you can enhance and change a business application quickly and easily.

Creating components also helps in reusing the business logic. By reusing well-designed business logic, an application can be opened up to different client channels with minimum effort, and no rework of the existing server components.

The CICS API provides several facilities to simplify the separation of application logic into components. For example, you can use the COMMAREAS and the **`LINK`** and **`XCTL`** commands.

Error handling

Error handling in an application is important. Categorize errors into the following types, and decide in advance how to handle each type of error:

- Conditions that are not normal from a TXSeries point of view, but are anticipated in program
- Conditions caused by user and input data errors
- Conditions caused by omissions or errors in application code
- Errors caused by mismatches between applications and TXSeries resources, for example, when a file is not found
- Errors relating to hardware or other system conditions beyond application program control

Common data structure

It is a good idea to store common structures in a shared library, so that multiple programs can access the structure. It also eases program maintenance and management.

Testing

Testing is an important phase of the application development cycle. You should always perform rigorous stress testing with good code coverage before deploying the applications to production systems.

Data choices

When *recoverable* resources, such as files and queues, are updated, extra logging is required, which incurs more input/output (I/O) and processor use. Resources defined as *unrecoverable* incur less logging. Because of this, define resources as recoverable only if they really must be.

Short logical units of work

It is important to keep ACID transactions as small as possible in terms of time. Longer transactions mean longer LUW that increase the duration of locks on recoverable resources. Locks can be for storage, updated data, and so on. A good example of a long LUW is a conversational program, where the LUW extends for the duration of the conversation. Therefore, all data is locked for the duration of the conversation.

The longer the locks are held, the greater the possibility of another transaction wanting access to the same resource, which leads to a wait situation. Therefore, it is important to keep LUWs as short as possible.

Using pseudo-conversational programs

Programs that interact with a terminal-based user are known as conversational. Any resources updated early in the conversation will require locks to be in place while the program solicits input from a user. If a user takes time to respond, or even fails to respond, data previously updated will remain locked, possibly for extended periods of time.

To avoid this problem of locking data while waiting for user input, programs should use a technique called *pseudo-conversational programming*. At the end of every conversation with the user, the program terminates, updates any data, and releases data locks. The key here is that the program ends rather than waiting for user input. After the user inputs more data, the program restarts and continues processing.

Updating close to sync point

A sync point in a CICS transaction is the point at which updated resources are committed. After this point, the changes to the data cannot be undone or rolled back. After a resource is updated, the data is locked to prevent access by other transactions. Updating the resource as close as possible to the sync point, in terms of time, minimizes the possibility of locks affecting other users and transactions in the system.

Data conversion

TXSeries is an American Standard Code for Information Interchange (ASCII)-based system, where CICS Transaction Server is an Extended Binary Coded Decimal Interchange Code (EBCDIC)-based system. Therefore, data manipulated by both types of CICS can require data conversion as it is moved between systems. Though not difficult to set up, it is an extra resource cost and one that must be considered by application developers.

Choosing an operating system for development

Because TXSeries is supported on UNIX style systems (AIX, Solaris, Linux x86 and HP-UX on Itanium) and Microsoft Windows, there are multiple choices when it comes to deciding on a development environment. There are several factors to consider when choosing an OS for development. The following list describes some of these considerations:

- Number of physical machines required
- Operating system features, such as security and backup required by the administrators and users
- Availability of third-party software required by the organization
- Number and type of systems already deployed within the organization
- Whether developers require access to individual CICS systems for their own testing, or whether they can share systems

Due to the prevalence of existing machines, the choice of OS can be straightforward for some organizations. For others, it can be more difficult.

1.7.3 Administration choices

TXSeries provides system administrators several facilities for controlling and managing a TXSeries environment. Unlike the previous two sections, where a decision can have serious effects on the overall behavior and performance of the system, administration choices describe several facilities and functions available within TXSeries for helping to administer the system.

These facilities can be enabled or disabled as and when they are required, for example, to help diagnose a problem or provide some additional reporting.

Web-based Administration Console

TXSeries provides administrators a uniform and single view of all of the TXSeries regions, SFS, and PPC Gateway servers running on a system. With TXSeries Administration Console, the administrator can easily perform all of the TXSeries administration activities through an intuitive graphical user interface (GUI).

User exits

TXSeries provides a powerful customization option called *user exits*. A user exit (also referred to as a *user exit point*) is the point in a CICS program at which CICS can transfer control to a program that you wrote (a *user exit program*), and can then resume control when your program completes its work.

There are several points in CICS where a user program can be hooked, for example, task termination, sync point, and so on. By effectively implementing user exits, you can extend the functionality of TXSeries.

Transaction classes (TranClass)

You can control the number and priority of transactions running in a region at any one time by using transaction classes. Every transaction is associated with a transaction class.

Administrators can then define the number of transactions in a particular class allowed to run concurrently. Using TranClass is an effective way of making sure that high-priority transactions take precedence over lower-priority ones.

Number of application servers

TXSeries uses application server processes to run transactions. Administrators can control the minimum and maximum number of application servers available in a CICS region to process transactions. Therefore, administrators control the number of concurrent transactions that CICS is able to run.

The minimum and maximum numbers should be carefully tuned. If the maximum is very small, the processing time can be delayed, because the transactions will wait in the queue before it can execute. If the minimum number is too large, considerable system time and resources can be spent on managing the servers and their memory.

If the difference between the minimum and the maximum application servers is very large, then the initial transaction responses can be slow whenever there is any sudden surge in the transaction requests as the system will start creating application servers after the requests start arriving. The creation of new application servers are resource-intensive and time-intensive operations.

Monitoring and statistics

CICS provides statistics and monitoring tools that you can use to gain information about the running of a CICS system. Data provided by these tools can be used to provide information about any resource contention or other problems that can affect the performance of your CICS system.

You can tailor these tools to provide information at certain event points, or at certain time intervals in the running of the CICS system. You can also tailor them to specify the types of data about which CICS is to provide the information.

TXSeries V8.1 provides real-time monitoring capability by using Tivoli Monitoring Infrastructure.

Dump and trace

TXSeries provides many problem determination tools, including dump and trace utilities. There are two kinds of dumps:

- Transaction dump
- System dump

A transaction dump writes specified areas of memory to a file to assist you in debugging an application program, or to identify why an abnormal end of task (abend) or storage violation occurred. System dump gives a snapshot of the entire CICS region, with numerous details at the time of dump. This is useful in analyzing a system-wide problem.

Tracing can assist both application developers and system administrators, and it is often crucial when asking for product support. TXSeries divides trace data into two broad categories:

- Application trace
- System trace

Application trace refers to trace data generated from application-specific code, which programmers use to debug applications. System trace refers to traces generated from the CICS product itself, which system administrators and product support staff use to analyze and diagnose system-wide problems.

1.8 Conclusion

TXSeries for Multiplatforms is part of the industry-leading CICS Family of products by IBM. It is production proven for over two decades in providing a powerful online transaction processing environment for the distributed platform. With the current TXSeries for Multiplatforms, IBM offers the next generation of distributed CICS transaction servers.

TXSeries software is designed to help you deploy application and integration programs that are written in enterprise programming languages, such as COBOL, C, C++, PL/I, and Java. By providing services that interact with the underlying hardware and software, TXSeries hides the complexity of your information technology (IT) systems without compromising their functionality.

TXSeries services enable enterprise developers to concentrate on solving tangible business problems with innovative technical solutions. It delivers a transactional online and batch processing environment that supports critical applications written in enterprise programming languages.

The OLTP environment supports the base-level CICS API. The integrated CICS SFS delivers a record-orientation file system. TXSeries for Multiplatforms is ideally suited for deployment in two common scenarios.

Due to its extensive support for many EIS, such as CICS Transaction Server, IMS, DB2, WebSphere MQ, and WebSphere Application Server, TXSeries is commonly used as a rapid deployment transactional integration server. Also, due to its wide range of support for enterprise programming languages and data access services, combined with the ability of TXSeries to scale up to CICS TS on z/OS, it is commonly used as an entry-level transaction server.

When comparing DTP solutions to mainframe transaction processing solutions, mainframes remain the undisputed leader in terms of both QoS and functional capability. However, distributed transaction processors are the perfect solution for smaller-scale, business-critical deployments. DTP solutions are also well-suited for highly distributed applications, where the lower service level agreements (SLAs) that come with a more open, decentralized environment are acceptable.

One of the largest advantages of TXSeries for Multiplatforms is that clients have a platform for the future. TXSeries provides interoperation with the most recent IBM WebSphere and mainframe technologies. It can also scale up to the world's leading mainframe transaction processing solution, CICS Transaction Server for z/OS, if required.

2

IBM TXSeries environment

In this chapter, we present a complete picture of a typical distributed environment involving IBM TXSeries. We also provide an overview of the different components involved in this TXSeries environment.

This chapter provides information about the following topics:

2.1 TXSeries environment

Figure 2-1 shows a complete picture of a TXSeries environment, where different client systems and resources that interact with TXSeries are illustrated.

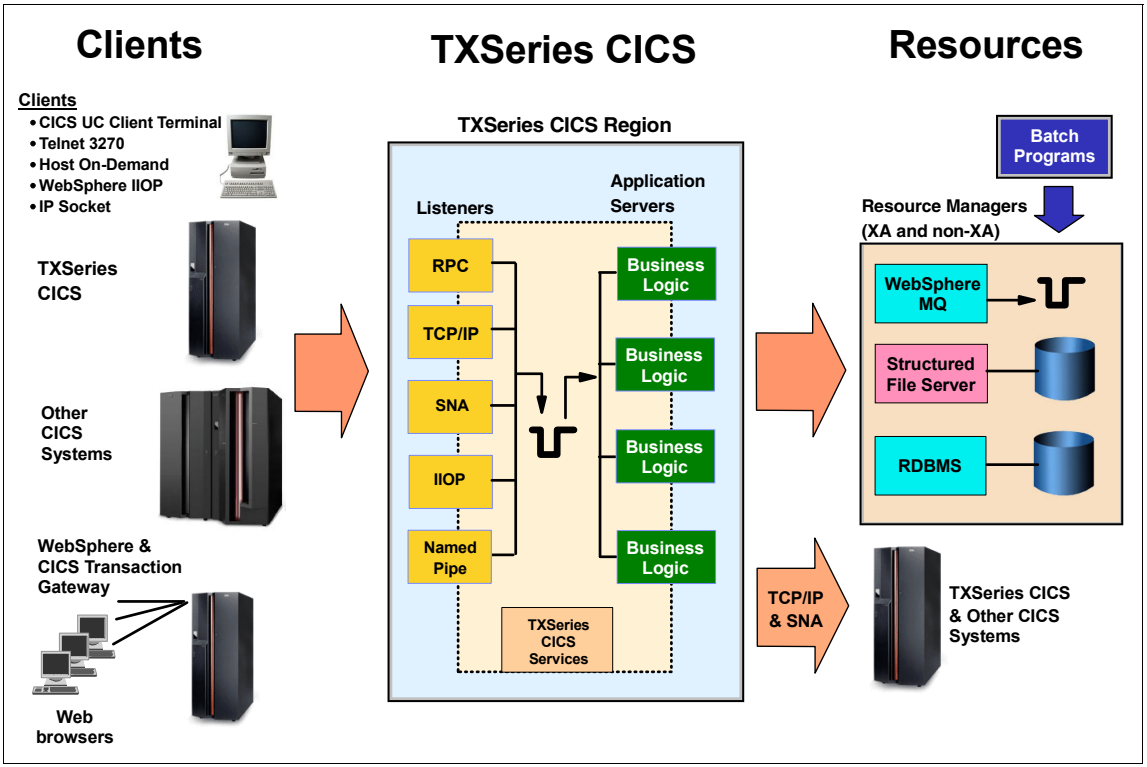


Figure 2-1 The complete picture

2.2 TXSeries region

The core of a TXSeries system is a *TXSeries region* (also referred to as an IBM CICS region or simply a region). In a TXSeries environment, the region provides all the functionality required as a transaction processing system. All the business logic applications are initiated and managed within the scope of a region.

The applications must be defined to a TXSeries region before their services can be consumed. A TXSeries region consists of a set of processes on an operating system (OS). The processes mainly include application servers where the actual business logic module gets run, and the listener processes that support various communication methods to interact with the region.

2.3 TXSeries clients

TXSeries clients act as a front-end application to access resources managed by a CICS region on TXSeries. They enable users to access regions for running their business applications.

Typically, the clients run on a different computer from the server, either because there is a need to connect to the server from different locations, such as automated teller machines (ATMs), or to offload the processor usage from the server machine. Because the client and the server are two different entities, they use a standard network for communication purposes. Multiple clients can simultaneously access CICS region.

Figure 2-2 shows the CICS client and the CICS server network.

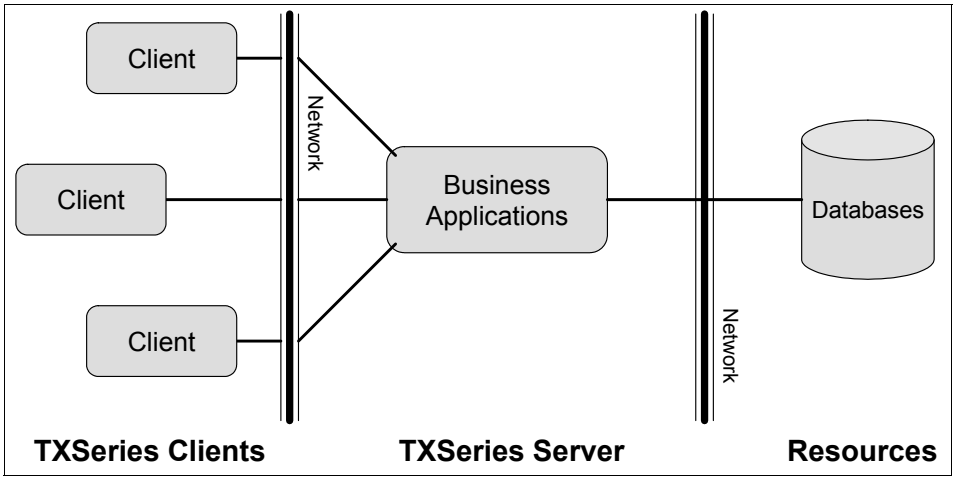


Figure 2-2 CICS client and CICS server

The client logic is typically isolated from the business logic that runs on the server machine. This isolation gives you enormous flexibility in the type of user interface (UI) that you decide to implement across different hardware types and operating systems.

The simplest way to access a region is to use a 3270-based terminal emulator called `cicslterm`, which is supplied with the product. A TXSeries region supports connection through different types of clients:

- TN3270. A region can be connected through a TN3270 client, such as `cicslterm`, or any external TN3270 client, such as IBM Personal Communications or IBM CICS Transaction Gateway (CICS TG) client.
- External call interface (ECI). ECI is a non-3270 interface provided by the CICS TG product. A client can be linked to ECI libraries to connect to a region. You can create your own front-end interface, and combine it with the ECI libraries to provide the CICS client component.
- External presentation interface (EPI). EPI enables a user application to act as a logical 3270 terminal, and to exchange the 3270 data stream with a CICS application.
- Web services. TXSeries applications can be made available as web services. Any web service client can start the business logic application managed in a TXSeries environment.

  The IP socket client establishes a connection with the listener through socket operations. The listener accepts the connection from the client, and gets the socket descriptor. It uses **`takesocket`** and **`givesocket`** mechanisms for communication.

The decision about which CICS client to use might largely depend on the existing hardware and OS on the desktop of the user. Also, you have to review the functions provided by each type of CICS client, cross-check them with the needs of the proposed CICS system or application, and then select the best CICS client. We describe some of the commonly used client deployment models in detail in 3.4, “Client deployment models” on page 51.

2.4 File Manager

In TXSeries, a resource can be accessed as a file similar to Virtual Storage Access Method (VSAM) file access by CICS on z/OS. A region in a TXSeries environment provides the necessary file services through a File Manager. Access to different file formats, such as key-sequenced data set (KSDS), entry-sequenced data set (ESDS), or relative record data set (RRDS), is supported by TXSeries.

For a TXSeries region to function, it must be associated with a File Manager. Usually, the Structured File Server (SFS), which is a component in TXSeries, is used as a File Manager to provide the file emulation and queuing services in a region.

Alternatively, an IBM DB2 or Oracle database can be configured as a File Manager for a region. File Manager tools included in the product, such as `cicssdt`, `cicsddt`, and `cicsodt`, can be used to create KSDS, ESDS, or RRDS files.

2.5 Resource manager access

Data in a transactional system is usually managed in a database or a messaging system, such as IBM WebSphere MQ. In a TXSeries environment, the databases and WebSphere MQ that are defined and managed by a region are referred to as *resource managers* (RMs). A TXSeries region enables transactional access to external RMs for database and messaging systems.

IBM TXSeries for Multiplatforms provides a transactional integration point for data and messaging, as illustrated in Figure 2-3. Full data integrity is provided through support of the extended architecture (XA) interface, the industry-standard interface defined by X/Open for managing commitment and recovery of transactional data, including the two-phase commit process.

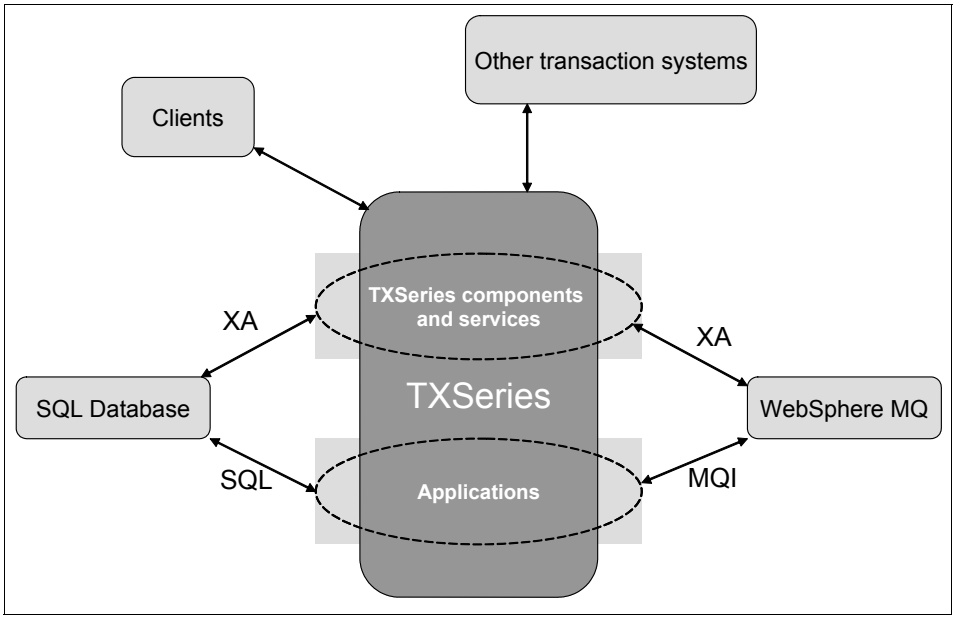


Figure 2-3 Transactional integration point

The following XA technology-compliant database management and messaging systems are supported by TXSeries for Multiplatforms:

- DB2
- IBM Informix
- Microsoft SQL Server
- Oracle
- Sybase
- WebSphere MQ

TXSeries uses the Transaction Manager-XA Service (TM-XA) interface to map transaction events to XA calls. TM-XA translates all information about transaction context and scope into appropriate XA calls that it sends to the XA resource manager (database). It uses the XA protocol to communicate transaction `prepare`, `commit`, and `end` states to RMs involved in those transactions.

Application programs running on a CICS server can use embedded Structured Query Language (SQL) calls to access data in relational database management system (RDBMS) databases. An SQL application program must be precompiled with an appropriate precompiler provided by the database, and linked to appropriate database libraries. It generally needs some way of defining which database the program's SQL calls are to be used with.

Application programs can use the WebSphere MQ Client application programming interfaces (APIs) for accessing queues in WebSphere MQ. In an environment where the RMs are configured to a region, the connection to the RMs is maintained by the region. There must not be any explicit statement in the application to connect or disconnect an active connection to the database managed by the region.

Apart from application programs accessing RMs through the XA protocol, they can also establish access to the RM directly, which is called a *non-XA connection*. The application programs in such cases must establish and maintain connection to the RM.

The programs issue SQL calls for accessing RDBMS databases, or WebSphere MQ Client APIs for accessing queues in WebSphere MQ. TXSeries for such RM connections does not issue any transactional updates, and requires the application to maintain its transactional integrity with the RM.

RMs also provide certain optimization features, which can enhance the performance of the transactional system, such as read-only optimization and one-phase commit.

In a two-phase commit configuration, there are two phases that are involved during the conclusion of a transaction:

- The prepare phase
- The commit phase

In a *prepare* phase, the region gets confirmation from all of its participants (RMs and file managers) on its readiness to commit, and then a *commit* is issued after getting a confirmation. A region can be configured to use a one-phase commit process, in which the prepare phase gets bypassed. We suggest that you use the one-phase commit optimization if you have only one RM to be accessed in your transactional system.

Another optimization refers to a technique called *dynamic registration*. Certain RMs provide this technique, which reduces the overhead for infrequently used RMs in the applications.

See 3.5, “Overview of resource managers” on page 67 for details about access of the supported RMs in a TXSeries environment.

2.6 Access to other CICS systems

TXSeries for Multiplatforms provides interoperability capabilities that enable it to play key roles in integrating applications in a heterogeneous environment. It offers comprehensive interoperability with mainframe systems, such as IBM CICS Transaction Server (CICS TS) and IBM IMS products.

TXSeries for Multiplatforms can act as a transaction server running on an intermediate tier in a multitier environment, as illustrated in Figure 2-4 on page 42. It provides integration and intermediate business logic. In this scenario, the back-end enterprise systems to which TXSeries for Multiplatforms can be connected include IBM CICS Transaction Server for z/OS, IMS, and other TXSeries systems.

Figure 2-4 shows TXSeries for Multiplatforms in a multitier environment.

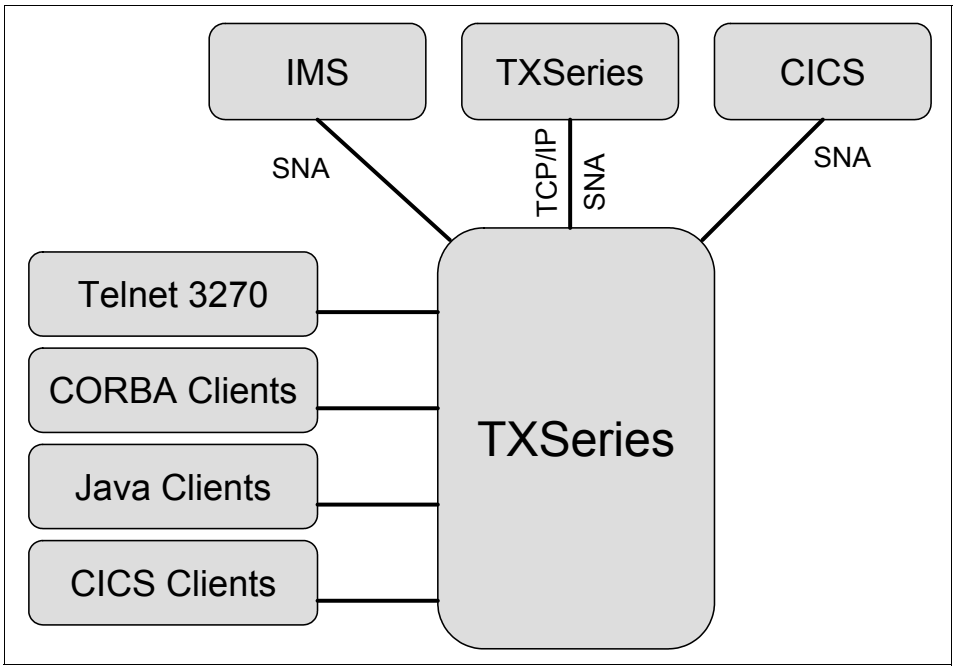


Figure 2-4 Transaction server running on an intermediate tier in a multitier environment

TXSeries CICS can communicate with remote CICS systems, IMS, and other TXSeries systems. The intercommunication with other remote CICS systems and TXSeries systems are supported over Systems Network Architecture (SNA) or Transmission Control Protocol/Internet Protocol (TCP/IP). With high data integrity through synchronization level 2 (SL2), the intercommunication facilities permit the following actions:

- CICS file and queue sharing
- Remote program linking
- Transaction routing
- Remote asynchronous processing
- Distributed transaction processing (DTP)

When designing such intersystem communications, the choice of communication protocol is important. The following communication protocols are supported:

- CICS Family TCP/IP support enables connectivity to TXSeries regions and CICS Transaction Gateway Clients.
- Peer-to-Peer Communications (PPC) TCP/IP support enables connectivity between TXSeries regions.

- Internet Protocol interconnectivity (IPIC) over TCP/IP support enables connectivity to CICS Transaction Gateway Clients, TXSeries regions, and other CICS systems.
- Local SNA support provides the fastest SNA connectivity offered by CICS. It enables TXSeries applications to communicate with every other member of the CICS Family.
- PPC Gateway SNA support. TXSeries uses TCP/IP to communicate with the PPC Gateway server, and the PPC Gateway server provides a link to the SNA network. The PPC Gateway server is required to be on the same machine as the CICS region.

Each of these five options has several advantages and disadvantages. It is important that these are considered carefully when deciding which communication option to use.

Table 2-1 summarizes the communication methods that TXSeries can use.

Table 2-1 TXSeries: Communication methods

Communication method	Best for	Restrictions
CICS Family TCP/IP	Communicating at SL0 or SL1 with TXSeries and CICS TG Clients.	DTP is not supported.
PPC TCP/IP	Communicating at SL0, SL1, or SL2 with other TXSeries regions.	Communication across machines can be supported by setting the `CICS_HOSTS` environment variable.
IPIC over TCP/IP	Communicating at SL0 or SL1 with other TXSeries regions, IBM CICS Transaction Gateway Clients, and IBM CICS systems. Also, ensuring secure communication by enabling Secure Sockets Layer (SSL) support.	Only distributed program link (DPL) is supported.
Local SNA	Fast SL0 or SL1 communication with remote logical unit 6.2 (LU 6.2) Advanced Program-to-Program Communication (APPC) systems. These connections can be used to connect to any CICS product.	Install a supported SNA product (for example, IBM Communications Server) on the same machine as the CICS region.
PPC Gateway SNA	SL0, SL1, and SL2 communication with remote LU 6.2 (APPC) systems. These connections can be used to connect to any CICS product.	Install a supported SNA product (for example, IBM Communications Server) on the same machine as the PPC Gateway and the CICS region.

It is also important to understand the differences of the character sets that are used by TXSeries and other CICS systems. TXSeries CICS uses the American Standard Code for Information Interchange (ASCII) character set. If you are connecting to a CICS system that uses the Extended Binary Coded Decimal Interchange Code (EBCDIC) character set, such as CICS TS for z/OS, you have to convert the data, either application-specific or TXSeries resource names.

There are two distinct areas where data conversion becomes an important consideration:

- If the application logic assumes certain collating or sorting sequences, or performs character bit manipulation, you might have to change this logic as part of the conversion from EBCDIC to ASCII. For example, in ASCII, you can convert a character from uppercase to lowercase by adding 32 to the binary value of the character.

  For EBCDIC, the same calculation requires a subtraction of 64 from the binary value. There is limited support from the CICS API for such program logic, and such issues might only be discovered through application testing. A possible solution is that most Common Business Oriented Language (COBOL) compilers can modify the sorting sequence behavior through a compiler directive.

- If TXSeries systems and other EBCDIC-based CICS systems have to share data or communicate with each other, some technique for converting the data between ASCII and EBCDIC is required. Both CICS systems provide formal methods for data conversion based on conversion templates. If these methods are not used, the application programmers can, of course, use their own developed techniques.

  Whenever both CICS systems communicate, there are two categories of data that might require conversion:

  - The name of the CICS resource. CICS resource names must always flow across the link in EBCDIC. The system that is not typically using EBCDIC is responsible for the translation. However, this translation is an internal CICS function, and does not require user setup.
  - The application data. If this requires conversion, it must be converted by the system that owns the resource. For example, if CICS TS for z/OS sends a request to read data from a file owned by TXSeries system, TXSeries is responsible for converting the ASCII data read from the file into EBCDIC before passing to CICS TS for z/OS.

    You must define suitable conversion templates, and no application programming is required. If application data passed through channels and containers, the data can be converted automatically by CICS using appropriate options in the API commands.

Extensive details about data conversion, including when it takes place, who is responsible, and how to configure it, are discussed in the following resources:

- The Communicating task under the relevant product version in the TXSeries IBM Knowledge Center:

  http://www.ibm.com/support/knowledgecenter/SSAL2T/welcome
- Chapter 7, “Upgrading to the latest version of TXSeries” on page 359
- Appendix B, “Additional material” on page 485

Detailed instructions about the integration of TXSeries systems with the CICS host are presented in 4.3, “TXSeries and CICS host” on page 128.

2.7 Conclusion

One of the key roles that probably any middleware platform drives is to coexist with other enterprise-wide business applications. TXSeries enables integration of enterprise-wide business applications and development of new business-critical applications. TXSeries provides interoperability capabilities that enable it to play key roles in integrating applications in a heterogeneous environment.

The following list describes some of the ways in which TXSeries can connect users and client applications:

- From UNIX and other desktops using a Telnet 3270 client
- From UNIX and other desktops through a CICS Transaction Gateway client
- From a web browser through a CICS TG
- From any Java environment through a CICS TG
- From a web services client

TXSeries provides methods to integrate with other enterprise applications, such as database (RDBMS) systems, messaging systems, other CICS or TXSeries systems, systems not based on CICS, and security. Chapter 4, “Enterprise integration” on page 117 provides details about how TXSeries can access different enterprise systems.

3

Developing distributed IBM CICS applications

In this chapter, we outline the use of development systems to produce applications for use with IBM Customer Information Control System (IBM CICS) distributed applications. We describe two development scenarios, a *lightweight* Microsoft Windows system, and a *heavyweight* UNIX system. In both cases, we document how we prepared the system and the settings that we used. This chapter explains the development of the *BigBlueBank* sample application that we developed to demonstrate many of the capabilities of IBM TXSeries.

We describe the development of object-oriented programs for TXSeries. We also describe different client deployment models, and the development of programs to suit these models. We demonstrate the integration of TXSeries applications with resource managers (IBM WebSphere MQ, IBM DB2, Oracle, and an IBM z/OS CICS system). We also examine the development of offline batch applications dealing with data that is created and managed by TXSeries.

This chapter provides information about the following topics:

3.1 Preparation

In this chapter, we use the following software components:

- IBM TXSeries for Multiplatforms
- WebSphere MQ
- DB2
- Oracle
- IBM CICS Transaction Gateway
- IBM Common Business Oriented Language (COBOL) for AIX
- Microsoft Developer Studio .NET

3.1.1 The CICS region

We create a CICS region called *DemoApp* and set an environment variable called **CICSREGION** to refer to it:

```
cicscp -v create region DemoApp
```

Windows:

```
SET CICSREGION=DemoApp
```

IBM AIX:

```
CICSREGION=DemoApp
export CICSREGION
```

3.2 Development environment for Windows

In this section, we describe a lightweight minimal development environment. We use development tools on Windows, together with TXSeries for Windows.

3.2.1 Language environment

The following sections describe the various compiler tools on Windows that we used in the development environment.

C

We used Microsoft Visual Studio .NET on Windows as a C development environment. Consult the Microsoft documentation for details about the installation and customization of this development environment.

IBM COBOL

IBM WebSphere Developer for z/Series and IBM COBOL for Windows are included as part of the IBM Rational Application Developer for zSeries. Insert disk 1 of IBM Rational Application Developer for zSeries and follow the on-screen prompts.

The BigBlueBank system can also be run using IBM COBOL for Windows, but this process is not described further in this book. We provide IBM COBOL sources and copybooks in the `Sources` and `copybooks` directories:

1. Use the compile_all script to compile the programs. Alternatively, you can choose to do so by setting the following environment variables:

   ```
   SET COBCPYEXT=CPY
   SET SYSLIB=%BANK%/Copybooks
   ```

2. Use the following command:

   ```
   cicstcl -lIBMCOB <sourcefile>
   ```

   For example, if the source file is `MAINMENU.cpp`, use the following command:

   ```
   cicstcl -lIBMCOB MAINMENU.ccp
   ```

3. Copy the object module to the region's `bin` directory:

   ```
   COPY MAINMENU.ibmcob %REGION%/bin
   ```

3.2.2 Resource manager

We used DB2 and Oracle as resource managers in an extended architecture (XA) environment. We further describe the configuration details for these resource managers in 3.5, "Overview of resource managers" on page 67.

3.2.3 Client

For a CICS client, we used CICS local terminal, called *cicslterm*, which is a 3270-based terminal emulator. No specific configuration is required other than what is supplied along with the TXSeries product. More client deployment models are further discussed in 3.4, "Client deployment models" on page 51.

3.2.4 System configuration: Networking

Because this environment is designed to be run on a Thinkpad, which might not have a permanent network connection, you might want to consider installation of the Microsoft *loopback* network connection. For details of the installation and configuration of the loopback adapter, see the following website:

http://support.microsoft.com/kb/2777200

3.3 Development environment for UNIX

In this section, we describe a lightweight minimal development environment. We use development tools on UNIX, together with TXSeries for UNIX.

3.3.1 Language environment

The following sections describe the various compiler tools on UNIX that we used in the development environment.

C

A C compiler is provided as part of the UNIX operating system (OS) software. To compile C programs that use the EXEC CICS construct, use the following command:

```
cicstcl -lC <program-name>
```

IBM COBOL

For details about the installation and customization of IBM COBOL for IBM AIX, consult the IBM COBOL documentation.

We provide IBM COBOL sources and copybooks in the Sources and copybooks directories:

1. Use the `compile_all` script to compile the programs. Alternatively, you can choose to do so by setting the following environment variables:

   ```
   export COBCPYEXT=CPY
   export SYSLIB=$BANK/Copybooks
   ```

2. Use the following command:

   ```
   cicstcl -lIBMCOB <sourcefile>
   ```

   For example, for the `MAINMENU.cpp` file, use the following command:

   ```
   cicstcl -lIBMCOB MAINMENU.ccp
   ```

3. Copy the object module to the region's bin directory:

   ```
   cp MAINMENU.ibmcob $REGION/bin
   ```

3.3.2 Resource manager

We used WebSphere MQ, DB2, and Oracle as resource managers in an XA environment. We further describe the configuration details for these resource managers in 3.5, “Overview of resource managers” on page 67.

3.3.3 Client

We used cicsterm, a 3270-based terminal emulator, as a CICS client. No specific configuration is required other than what is supplied with the TXSeries product. More client deployment models are discussed in the following section.

3.4 Client deployment models

In the following sections, we demonstrate different client deployment models for the sample BigBlueBank application.

3.4.1 3270 terminals

We provide samples of 3270-based programs that use basic mapping support (BMS) to provide a user interface (UI). See 8.2, “Installing the BigBlueBank sample application” on page 383 or details about the installation and configuration of this application between the subsystem and CICS terminal.

Running the 3270 programs

We run the programs by starting a 3270 terminal emulator for the region:

- AIX

  `/usr/lpp/cics/bin/cicslterm -r $CICSREGION`

  (Because of the **PATH** setting on our AIX login, we give the full path name to the emulator to ensure that we run the correct one.)

  On Windows, you need to configure NamedPipe listener to use `cicslterm`. See TXSeries library to configure NamedPipe listener for a region.

We are now ready to run the application:

1. Enter **BANK** transaction, and see the initial menu (Figure 3-1 on page 52).
2. Select the menu options.
3. To see a list of accounts, use option **4** to browse the file.

4. If WebSphere MQ, DB2, or both are enabled, produce output to them by choosing option **3** (Modify Details about an existing Customer) from the menu, as shown in Figure 3-1. Amend a record (DB2 output), or doing a credit/debit to an account (WebSphere MQ output).

```
BigBlueBank              CUSTOMER MAINTENANCE SYSTEM          Date: 11-23-05
 MENU  MAINMENU                     MAIN MENU                  Time: 11:59:56


      OPTION    ACTION TO BE TAKEN



         1      Add a New Customer

         2      Browse an Existing Customer

         3      Modify Details about an existing Customer

         4      Browse Details about ALL the Customers

         5      Get a Credit Rating for an existing Customer

         X      Exit the Customer Maintenance System

                Select the Action to be performed


 ENTER = Process the Option                                        PF3 = Exit
```

Figure 3-1 BigBank main window

3.4.2 CICS Transaction Gateway

Programs written in *traditional* languages, such as C and COBOL, can access the TXSeries environment using the CICS Transaction Gateway client. This enables such programs to use TXSeries resources.

CICS Transaction Gateway configuration

For details about the CICS Transaction Gateway configuration that we use in the BigBlueBank application, see 4.6.3, “Overview of CICS Transaction Gateway” on page 177.

Client side

In the sample application, there is a program called `CTGDemo.cbl` that demonstrates the use of the IBM CICS Transaction Gateway. It can be run on either UNIX or Windows.

Server side

On the server, update the program stanza for the CICS region in use to add the server-side programs. The example application uses a program. You can find the source code for this in `CUSTECI.ccp`. To add the server-side programs, follow these steps:

- For UNIX, use the following information:

  a. Set an environment variable to specify the location of the COBOL copybooks used by the application. Check the path name for the WebSphere MQ copy files, or omit it if you are not using WebSphere MQ:

  ```
  COPYPATH=$BANK/app/Copybooks:/usr/lpp/mqm/inc
  export COPYPATH
  ```

  b. Precompile the modules using the CICS precompiler. Then compile them using the IBM COBOL compiler. You can conveniently do this by using the **`cicstcl`** command:

  ```
  cicstcl -lIBMCOB CUSTECI.ccp
  ```

  > **Note:** A `compile_all` script is provided that compiles all the supplied programs. To use this script, you must first set an environment variable (**`BANK`**) to point to the directory where the sample application is installed:
  >
  > ```
  > BANK=/opt/bankapp
  > export BANK
  > ```
  >
  > The `compile_all` script prompts for the region name when it is run.

  c. Move the object files to the appropriate region directory:

  ```
  mv CUSTECI.ibmcob /var/cics_regions/$CICSREGION/bin
  ```

  d. Add the **`CUSTECI`** program to the program stanza for the region, as shown in Example 3-1.

  Example 3-1 Adding the CUSTECI program

  ```
  cicsadd -c pd -r $CICSREGION -P CUSTECI GroupName=""
  ActivateOnStartup=yes ResourceDescription="Customer ECI Program"
  AmendCounter=0 Permanent=no EnableStatus=enabled RemoteSysId=""
  RemoteName="" TransId="" RSLKey=public PathName="CUSTECI"
  ProgType=program UserExitNumber=0 TemplateDefined=no Resident=yes
  ```

  An **`entry.prog`** script is provided that adds all of the supplied programs to the program stanza. The script prompts for the region name when it is run.

- For Windows, use the following information:

  a. Set an environment variable to specify the location of the COBOL copybooks used by the application:

     ```
     SET COPYPATH= ..\Copybooks
     ```

  b. Precompile the modules using the CICS precompiler. Then compile them using the COBOL compiler. You can do this conveniently by using the **cicstcl** command.

  c. Copy the object files to the appropriate region directory.

     We provide a `compile_all.bat` batch file that compiles all of the supplied programs. To use this batch file, you must set a **BANK** environment variable to point to the directory where the sample application is installed:

     ```
     SET BANK=C:\opt\bankapp
     ```

     The `compile_all.bat` batch file prompts for the region name when it is run.

  d. Add the **CUSTECI** program to the program stanza for the region, as shown in Example 3-2.

     Example 3-2 Adding the CUSTECI program

     ```
     cicsadd -c pd -r %CICSREGION% -P CUSTECI GroupName=""
     ActivateOnStartup=yes ResourceDescription="Customer ECI Program"
     AmendCounter=0 Permanent=no EnableStatus=enabled RemoteSysId=""
     RemoteName="" TransId="" RSLKey=public PathName="CUSTECI"
     ProgType=program UserExitNumber=0 TemplateDefined=no Resident=yes
     ```

  We provide an **entry.prog.bat** script that adds all of the supplied programs to the program stanza. The script prompts for the region name when it is run.

> **Tip:** Remember that if you make subsequent changes to the programs in a running region, you should use the **CEMT** transaction to renew the image, as shown in the following example:
>
> ```
> CEMT SET PROG (Program) NEW
> ```

3.4.3 Java clients

We provide connectivity for Java programs through WebSphere, so that the programs can benefit from the features and capabilities of WebSphere. The installation, customization, and programming of this part of the system is detailed in “Chapter 4, “Enterprise integration” on page 117.

3.4.4 Secured sessions for TCP/IP clients

We show you various client deployment models possible with TXSeries that access TXSeries regions through the CICS Transaction Gateway product. The connection between these two products is generally established through a Transmission Control Protocol/Internet Protocol (TCP/IP) link. However, by default this is not secure. This means that the data that is transmitted back and forth from client and server remains unencrypted.

Therefore, to establish a secure channel between a TCP/IP client and TXSeries regions, one of the methods that we use is the Secure Shell (SSH) protocol. The SSH protocol encrypts all of the traffic between the client and the server. We use OpenSSH on AIX to demonstrate how to use the SSH protocol with TXSeries.

The SSH protocol provides a facility known as *local port forwarding*, which you can effectively use for creating a secure channel. There are three primary pieces of information to configure SSH local port forwarding:

- A local unused port (on the client machine)
- The host name of the server machine
- The server's port that the client wants to connect to

The SSH protocol establishes a secure channel from the specified local port (on the client machine) to a known remote port (on the server machine). The CICS Transaction Gateway product, or any other TCP/IP client, sends the data to the local port (on the client machine). The data is then encrypted by SSH and forwarded across to the remote port (on the server machine).

Therefore, a secure channel is established between the client and the TXSeries server. Figure 3-2 illustrates this scenario.

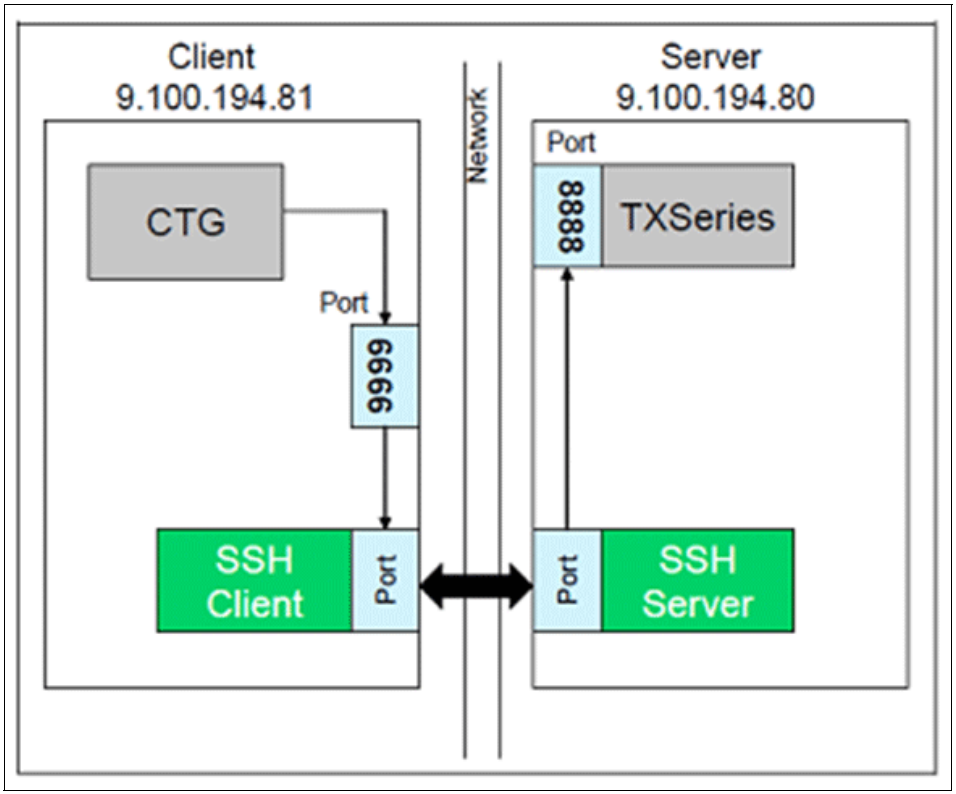


Figure 3-2 Secure channel established between the client and TXSeries

We begin by configuring SSH on both the client and the server machines.

Note: We use the OpenSSH that comes bundled with the AIX operating system. For demonstration purposes in this book, both the client and server machines used run on AIX.

Configuring the server side (9.100.194.80)

The server is the machine or host on which the TXSeries regions are running. The following steps demonstrate how to configure SSH on the server:

1. OpenSSH is bundled by default with UNIX. If it is not present, you must download and install the appropriate SSH software on your server machine.

   OpenSSH, by default, is installed in the `/usr/local/bin` directory. The configuration files are stored in `/usr/local/etc`, and the SSH server is in the `/usr/local/sbin` directory.

2. Ensure that you have the host key for the server machine. This is present under the SSH configuration directory (`/etc/ssh`). The following files must be present in the configuration directory:

   - `ssh_host_key`
   - `ssh_host_rsa_key`
   - `ssh_host_dsa_key`

   If these key files are not present, you can generate them with the following commands:

   - `ssh-keygen -t rsa1 -f /etc/ssh/ssh_host_key -N ""`
   - `ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key -N ""`
   - `ssh-keygen -t dsa -f /etc/ssh/ssh_host_rsa_key -N ""`

3. Enable the port forwarding option in the server configuration file by enabling the **`AllowTcpForwarding`** option with the value `yes`.

   > **Tip:** You can tune the port forwarding mechanism to enable it for selected users. See the SSH documentation for more information about this.

4. Start the SSH server daemon by entering the following command:

   ```
   /usr/sbin/sshd &
   ```

   When the SSH server daemon starts successfully, you have configured the server side.

Configuring the client side (9.100.194.81)

We use the CICS Transaction Gateway product as the TCP/IP client to talk to TXSeries region using a secured channel. However, you can use another TCP/IP client to have a secure channel in a similar way.

The CICS Transaction Gateway client configuration file (`CTG.INI`) must have an entry about the server region.

Example 3-3 shows an example of one such entry in the `CTG.INI` client configuration file.

Example 3-3 CICS Transaction Gateway client configuration file server region entry

```
SECTION SERVER = BANKREG
DESCRIPTION=BANKREG TXSeries Region
UPPERCASESECURITY=N
PROTOCOL=TCPIP
NETNAME=pons1
PORT=1435
CONNECTTIMEOUT=0
TCPKEEPALIVE=N
ENDSECTION
```

In Example 3-3, the **`NETNAME`** specifies the host name on which the regions exist (PONS1), the **`PROTOCOL`** specifies the TCP/IP protocol to be used (TCPIP), and the **`PORT`** is the server TCP/IP port on the server machine (`1435`).

To configure SSH on the client machine, perform the following steps:

1. Select an unused port number on the client machine. We use the port `6666` for this example.
2. Determine the server machine host name and the server port. We use a server machine IP of `9.100.194.80` and a server port of `8888` for this example.
3. Run the following SSH command on the client machine:

   ```
   /usr/bin/ssh -L6666:localhost:8888 9.100.194.80
   ```

4. After you run the command, the SSH displays a message (Example 3-4).

   Example 3-4 SSH message

   ```
   {pons1:root}/etc/ssh -> which ssh
   /usr/bin/ssh
   {pons1:root}/etc/ssh -> ssh -L6666:localhost:8888 9.100.194.80
   The authenticity of host '9.100.194.80 (9.100.194.80)' can't be
   established.

   RSA key fingerprint is
   ec:b2:84:bf:87:c7:52:3b:ef:b2:b8:71:41:09:08:4c.
   Are you sure you want to continue connecting (yes/no)? yes
   Warning: Permanently added '9.100.194.80' (RSA) to the list of known
   hosts.
   root@9.100.194.80's password:
   ```

5. Answer `yes` to add the host name to its list of known hosts. SSH does the authentication. SSH provides different methods of authentication, such as password authentication, public key authentication, and so on.

   If you configure SSH for password authentication, you are prompted by a login prompt where you have to enter the user ID and password. When the password authentication is successful, a secure channel is established between the client and the server machine.

   > **Important**: The local port forwarding mechanism is closed when you close the session. Instead, you can use SSH2 protocol. Therefore, even if the session is closed, the local port forwarding mechanism works in the background.

6. Modify the `CTG.INI` configuration file to reflect the local port number on the client machine, rather than the remote port of the server machine. Example 3-5 shows the updated client configuration file.

   Example 3-5 Updated client configuration file

   ```
   SECTION SERVER = BANKREG
   DESCRIPTION=BANKREG TXSeries Region
   UPPERCASESECURITY=N
   PROTOCOL=TCPIP
   NETNAME=pons1
   PORT=6666
   CONNECTTIMEOUT=0
   TCPKEEPALIVE=N
   ENDSECTION
   ```

7. Restart the client with the commands:

   - `cicscli /X=BANKREG`
   - `cicscli /S=BANKREG`

   You have now configured the client side.

8. With this setup, the CICS Transaction Gateway client sends all its data to the local port `6666` on the client machine. This is received by the SSH protocol where it encrypts the data and sends it to the server. At the server side, the SSH daemon receives the encrypted data, decrypts it, and forwards it to the server port `8888` on the server machine.

   You might have to configure the SSH to fine-tune it to your security requirements. See the SSH documentation for more information about how to configure SSH. Talk to your system administrator before you enable the local port forwarding mechanism of SSH.

3.4.5 Telnet clients (TN3270)

TN3270 is a protocol to emulate IBM 3270 terminal display over an IP network. This was originally defined in Request for Comments (RFC) 1576, and is based on the Telnet protocol. The basic difference between a normal Telnet session and a TN3270 session which character set they use.

Telnet uses the American Standard Code for Information Interchange (ASCII) character set and sends a line of data at a time. The latter uses the Extended Binary Coded Decimal Interchange Code (EBCDIC) character set and sends a block of data at a time. The TN3270 is primarily enabled to connect to a 3270-based CICS system over an IP network.

The TN3270 communication has the following elements:

TN3270 client A front end that emulates the 3270 display terminal. The TN3270 client accesses a TN3270 server over an IP network.

TN3270 server A listener to a TN3270 client. The server converts the TN3270 data stream to Systems Network Architecture (SNA) 3270 data stream and passes the data to the TXSeries region. The TN3270 server is on the machine where the TXSeries region is running.

You can configure TXSeries to access CICS services from a TN3270-based client. This means that you can use Telnet to connect to the CICS region, in the same way as you do for other systems.

Figure 3-3 depicts the TN3270 client communication with TXSeries.

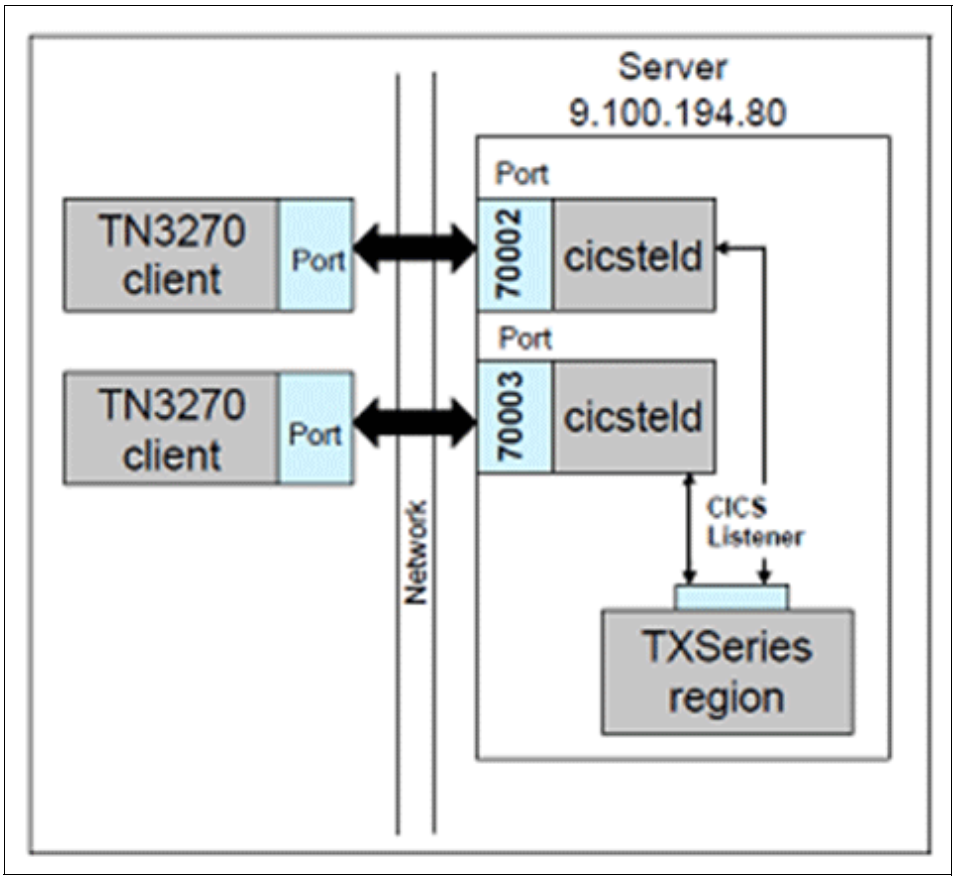


Figure 3-3 TN3270 client communication with TXSeries

TXSeries provides a TN3270 server called *cicsteld*, which establishes a Telnet session with a TN3270 client, and uses **`TERMINAL-TYPE`** negotiation for the Telnet option.

Requirement: Each TN3270 client session requires a dedicated TN3270 cicsteld server process to be running on the server machine.

Configuring TN3270 server on the region

To configure TN3270 server on a region, use the **cicscp create telnet_server** command, as shown in Example 3-6.

Example 3-6 Command to configure the TN3270 server

```
{pons1:root}/ ->
{pons1:root}/ -> cicscp -v create telnet_server tsrv1
ERZ096071I/0108: Creating a telnet server
ERZ096163I/0109: cicscp has assigned port number '70002' to telnet
server 'tsrv1'

ERZ096072I/0109: The telnet server 'tsrv1' was created successfully
ERZ096002I/0003: cicscp command completed successfully
{pons1:root}/ -> ls -l /var/cics_clients
total 16
-rwxr-x--- 1 cics cics 43 Nov 20 18:04 cicsteld.tsrv1
{pons1:root}/ ->
{pons1:root}/ -> cat /etc/inetd.conf | grep tsrv1
tsrv1 stream tcp nowait cics
/var/cics_clients/cicsteld.tsrv1
{pons1:root}/ ->
{pons1:root}/ -> cat /etc/services | grep tsrv1
tsrv1 70002/tcp
{pons1:root}/ ->
```

The **cicscp create telnet_server** command completes the following tasks:

1. Adds an entry in the operating system's `inetd.conf` file on AIX
2. Creates a script file under the `/var/cics_clients` directory
3. Creates an entry in the `/etc/services` file for the Telnet port

Note: You can also start the cicsteld server manually by passing command line parameters. The cicsteld session is ended when the TN3270 session is disconnected.

When a TN3270 client connects, the **inetd** process in the operating system automatically runs the shell script in the `/var/cics_clients` directory, thereby creating a TN3270 cicsteld server process. Then the cicsteld server process establishes the Telnet session with the TN3270 client, after negotiating the Telnet options.

Connecting from a TN3270 client

To connect from a TN3270 client, you have to enter the host name and the port number on which a TN3270 server is listening. When the client establishes a connection with the server, it lists all the available regions running on the server.

You select the region that you want to connect to, and run the transaction.

Unconfiguring TN3270 server for a region

To unconfigure the TN3270 server, you can run the following command:

```
cicscp -v destroy telnet_server <servername>
```

Here *<servername>* is the name of the TN3270 server.

Information: You can use the method explained in 3.4.4, “Secured sessions for TCP/IP clients” on page 55 to configure secure communication between TN3270 Telnet client and server. However this requires configuring SSH at each client machine, which might not be practical when thousands of clients connect to the region.

In these cases, the TN3270 client, such as IBM Personal Communications server, can be configured to access a host using an intermediate Communications Server for AIX node rather than accessing the host directly. The clients connect to the IBM Communications Server for AIX node that implements the TN Redirector component using TCP/IP.

The TN Redirector can establish a separate TCP/IP connection to cicsteld. The TN Redirector can use Secure Sockets Layer (SSL) support for authentication and encryption across Internet Protocol networks. It enables the use of SSL security checking where necessary between the Telnet client and the TN Redirector.

Code page conversions

Code page conversions are necessary because the TN3270 client and the CICS region operate in two different character sets. The TN3270 client uses the EBCDIC character set as supported by the Telnet protocol, and the CICS server uses the ASCII character set.

Therefore, it is mandatory for the cicsteld server process to convert from EBCDIC to ASCII when a request is received from the client, and to convert from ASCII to EBCDIC when a response is sent back to the client.

The cicsteld server process uses client code page `037` (EBCDIC) by default and `8859-1` (ASCII) as the default server code page. If you are using a different locale, you can change the default code page by specifying the **`-c clientCodePage`** and **`-s serverCodePage`** in the cicsteld server process.

Terminal mode emulation

The cicsteld server automatically determines the terminal mode to be used when communicating with TN3270 client. However, if you choose to use a user-defined terminal mode, you can override the default by specifying the **`-e emulation`** option in the cicsteld server process.

The following preferred CICS model types can be used by cicsteld:

- The `hft` and `mft` types for terminals with extended data stream
- The `lft` type for terminals that do not have extended data stream

Startup transaction

You can set up the cicsteld server process to run a startup transaction, after the connection is established with the TN3270 client. For example, you can run the CESN or CESL transaction as the first transaction to enable users to log on to your system. You can do this by specifying the transaction name using the **`-tinitialTransaction`** option in the cicsteld server process.

If the SIGNON fails, the user has default user access. To enforce SIGNON on connection and avoid default user access, you can set `CICS_SET_FORCE_SIGNON=1` in the region environment. Any region environment file changes require a restart of a region.

Using LUName option

The `LUName` option in the TN3270 client is used to represent a logical unit name connected over an SNA network. This option is available only for TN3270E-supported servers. However, TXSeries uses this field with the ability to represent an eight-letter character, which is usually used to identify the clients connected to the region.

To enable this, you must first turn on appropriate options to indicate to the client that it should use TN3270E services when supported. On the TXSeries regions side, you can access the value specified in this field (`LUName`) through the **`NETNAME`**.

Requirement: If you are using IBM Rational Host On-Demand TN3270 client, you must choose.

3.4.6 IP socket clients

You can use various client deployment models, which are described in this book, to access a CICS resource from a system not based on CICS. Many of the client deployment models either require certain product configurations or knowledge of well-defined programming interfaces, such as Advanced Program-to-Program Communication (APPC), external call interface (ECI), or external presentation interface (EPI).

In this case, we show another client that uses a fundamental network mechanism to talk to the TXSeries region. This is accomplished using *sockets*. If you are familiar with the sockets world, and you want flexibility in accessing a CICS resource from a system not based on CICS, you have the option of using IP to access a CICS resource.

IP socket clients require an associated server program, what we call a *listener program*, which listens and accepts connections from the IP clients. The listener program is a daemon process that does not end until you have to stop accepting connections from the clients. You can run this listener program in the following ways:

- As a stand-alone operating system process. This establishes a connection with the client, and starts a program in the CICS region using an external interface, such as ECI. This is illustrated in Figure 3-4 on page 66.
- As a CICS transaction. This establishes a connection with the client, and creates a new transaction using the **`EXEC CICS START TRAN`** command. This is illustrated in Figure 3-5 on page 67.

TXSeries provides sample implementation of **`givesocket()`** and **`takesocket()`** in C, which you can use to share a socket across CICS processes.

In Figure 3-4 on page 66, the IP socket client establishes a connection with the listener through socket operations. The listener accepts the connection from the client, and gets the socket descriptor.

The socket descriptor is shared among the CICS processes through the **`givesocket()`** mechanism, which is called by the listener. The listener starts the CICS program through an external interface, such as ECI, and through passing a communication area (COMMAREA), which contains information about the socket descriptor.

The CICS program then calls the **`takesocket()`** to get the socket descriptor that is associated with the client, and it starts a CICS transaction, passing the socket descriptor. The CICS transaction can then RETRIEVE the data, get the socket descriptor, and call send and receive systems to exchange data with the client directly.

In Figure 3-5 on page 67, the difference is that the listener program runs under the CICS region. After sharing the socket descriptor, the CICS listener program starts the CICS transaction, which retrieves the socket descriptor and establishes communication with the client directly.

Figure 3-4 illustrates the listener program as a stand-alone operating system process.

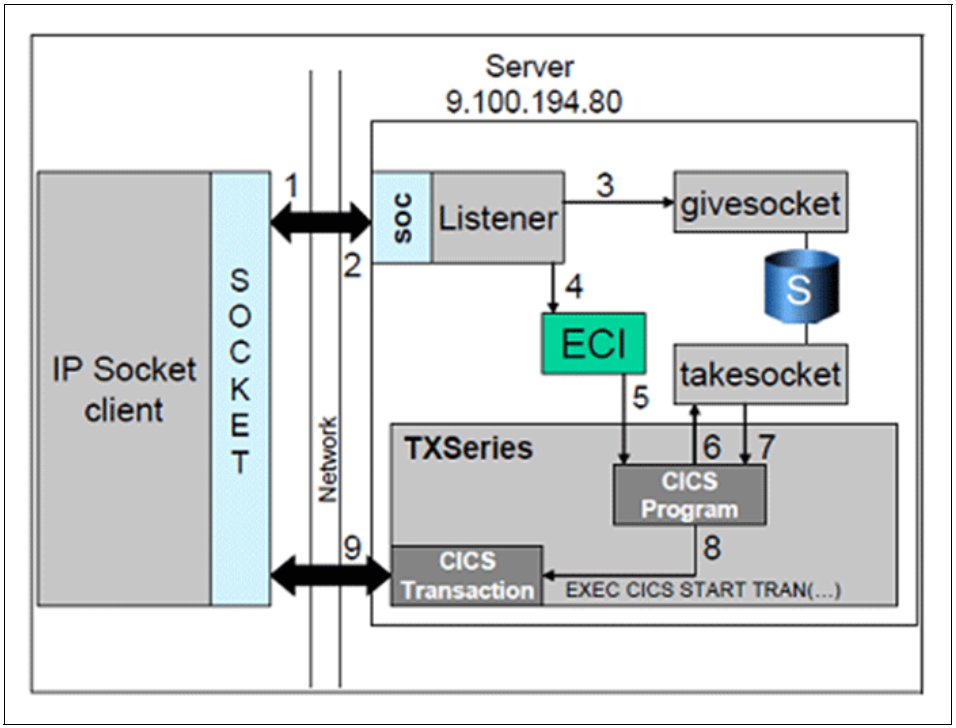


Figure 3-4 Listener program as a stand-alone OS process

Figure 3-5 illustrates the listener program running in a CICS region.

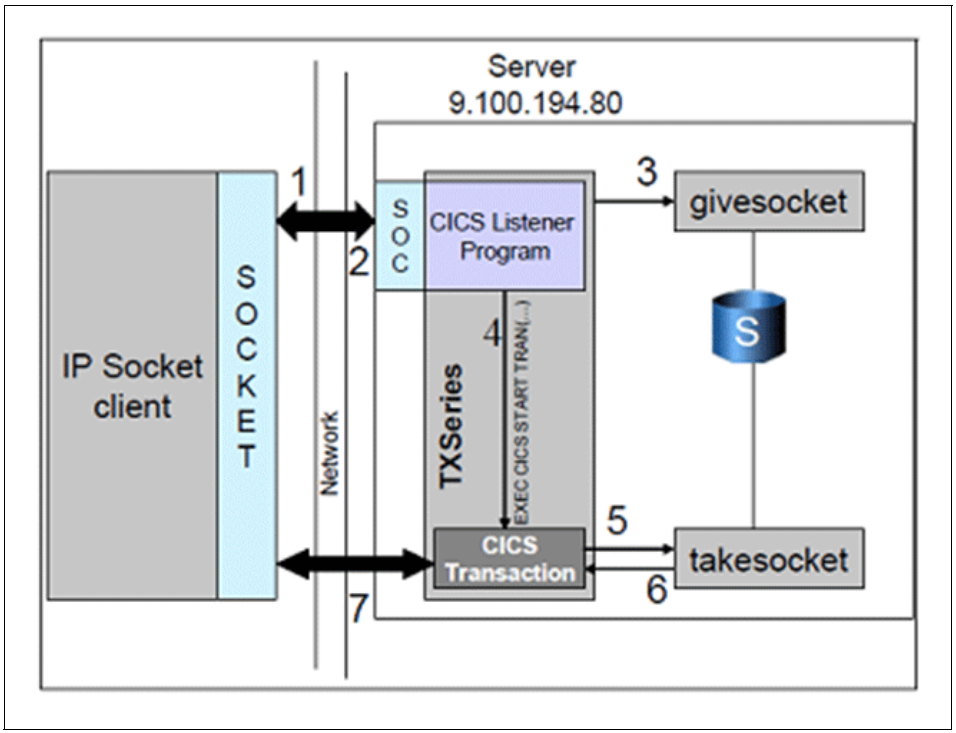


Figure 3-5 Listener program running in CICS

3.5 Overview of resource managers

In the following section, we present an overview of the various scenarios in which a resource manager (RM) can be accessed in a distributed transaction-processing environment. We also demonstrate different resource managers for the sample BigBlueBank application.

3.5.1 Resource managers in a distributed transaction processing model

Resource managers provide access to resources. Typically, we think of DB2 as a resource manager. Other resource managers are, for example, WebSphere MQ, other CICS systems, and other file systems. In a distributed transaction environment, the job of a resource manager is to provide access to the resource (database, message queue, and so on).

In addition, the resource manager can offer transactional capabilities to CICS. These capabilities might be offered through the X/Open XA specification. In the case of another CICS system, communication can be by the intersystem communication (ISC) protocol.

3.5.2 XA environment versus non-XA environment

For a full description of XA and non-XA connections in a CICS environment, see 4.1.1, “Configuring TXSeries for use with an RDBMS” on page 118. In our sample application, connection to WebSphere MQ and DB2 (or Oracle) is through an XA connection, and access to the Structured File Server (SFS) file system is a non-XA connection. This sample application also accesses DB2 using a non-XA connection for batch programs, where no transactional capability is necessary (Figure 3-6).

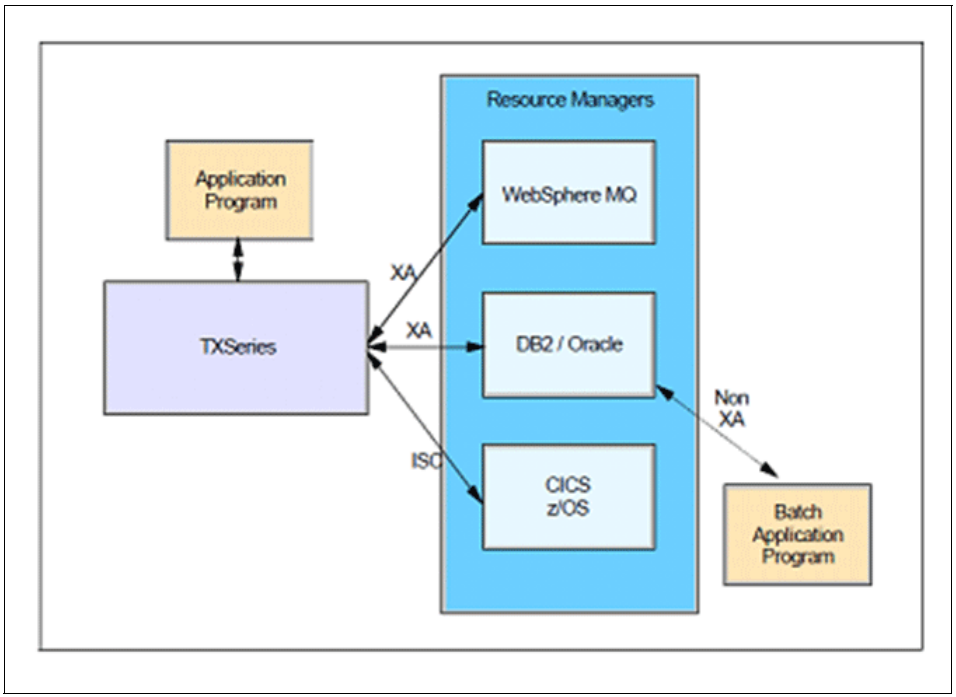


Figure 3-6 Sample application accessing DB2 using a non-XA connection

3.5.3 The switch-load file

For a description of the function of a switch-load file, see 4.1.1, “Configuring TXSeries for use with an RDBMS” on page 118. In this section, we describe the building of the switch-load file for each RM used by our sample application.

3.5.4 Configuring WebSphere MQ

In this section, we show you how to configure WebSphere MQ on both an AIX platform and a Windows platform.

AIX

To create the switch-load file, complete the following steps:

1. Use the commands shown in Example 3-7.

Example 3-7 Commands to create the switch-load file

```
echo "amqzscix" > tmp.exp
xlc_r /usr/mqm/samp/amqzscix.c -I/usr/lpp/cics/include
-I/usr/mqm/inc -e amqzscix -bE:tmp.exp -bM:SRE -o
/var/cics_regions/$CICSREGION/bin/amqzscix
/usr/lpp/cics/lib/regxa_swxa.o -L/usr/mqm/lib -L/usr/lpp/cics/lib
-lcicsrt –lEncina -lEncServer -lsarpc -lpthreads
-lmqmcics_r -lmqmxa_r -lmqzi_r -lmqmcs_r -lmqmzse
rmtmp.exp
```

Important: The WebSphere MQ commands that follow must be run by a user with appropriate permissions. On our demonstration system, this user is `mqm`. Therefore, before running these commands, we run the **`su-mqm`** command. You can choose to log in as the WebSphere MQ user rather than using **`su`**.

2. Create a queue manager called mqdemo:

   ```
   crtmqm mqdemo
   ```

3. Start the queue manager:

   ```
   strmqm mqdemo
   ```

4. Create a local queue. Create an input file called `mqdemo.in`, as shown in Example 3-8.

Example 3-8 Creating the mqdemo.in input file

```
DEFINE QLOCAL (CICS.LOCAL.QUEUE) +
DESCR('Local Queue') +
PUT(ENABLED) +
GET(ENABLED) +
PROCESS(CICS.PROCESS) +
REPLACE
```

5. Use this file as an input to the **runmqsc** command:

   ```
   runmqsc mqdemo < mqdemo.in
   ```

6. Add an entry for WebSphere MQ in the XA resource manager product definitions (XAD) stanza:

   ```
   cicsadd -c xad -r $CICSREGION MQXA ResourceDescription="MQXA Server"
   SwitchLoadFile=/var/cics_regions/$CICSREGION/bin/amqzscix
   XAOpen="mqdemo"
   ```

Windows

A switch-load file is provided in the `bin` directory of the WebSphere MQ Windows installation. It is called `mqmc4swi.dll`.

To create a new switch-load file, complete the following steps:

1. Change to the WebSphere MQ Windows sample directory and use the following command:

   ```
   nmake /f cics_mq_sample.mk switchload
   ```

2. Copy the created file to the region's `bin` directory:

   ```
   copy amqzscin.dll C:\var\cics_regions\%CICSREGION%\bin
   ```

3. Create a queue manager called `mqdemo`:

   ```
   crtmqm mqdemo
   ```

4. Start the queue manager:

   ```
   strmqm mqdemo
   ```

5. Create a local queue. Create an input file called `mqdemo.in`, as shown in Example 3-9.

Example 3-9 Creating the input file

```
DEFINE QLOCAL (CICS.LOCAL.QUEUE) +
DESCR('Local Queue') +
PUT(ENABLED) +
106 Revealed! The Next Generation of Distributed CICS
GET(ENABLED) +
PROCESS(CICS.PROCESS) +
REPLACE
```

6. Use this file as an input to the **runmqsc** command:

```
runmqsc mqdemo < mqdemo.in
```

7. Add an entry for WebSphere MQ in the XAD stanza:

```
cicsadd -c xad -r testreg WMQXA ResourceDescription="WebSphere
MQXA",SwitchLoadFile="amqzscin.dll",XAOpen="mqdemo"
```

3.5.5 WebSphere MQ and IBM COBOL

COBOL programs access WebSphere MQ using the **CALL** verb. This section describes how to configure COBOL programs on AIX and Windows.

Important: When you use the MQ*xxxx* routines from an IBM COBOL program, it is necessary to specify that the parameters should be passed **BY REFERENCE** or **BY VALUE**, as shown in the following code:

```
CALL 'MQCONN' USING QM-NAME, BY REFERENCE HCONN,
COMPLETION-CODE, REASON.

CALL 'MQOPEN' USING BY VALUE HCONN, BY REFERENCE OBJECT-DESCRIPTOR,
BY VALUE OPTIONS, BY REFERENCE Q-HANDLE,
OPEN-CODE, REASON.

CALL 'MQPUT' USING BY VALUE HCONN, BY VALUE Q-HANDLE,
BY REFERENCE MESSAGE-DESCRIPTOR,
BY REFERENCE OPTIONS,
BY VALUE BUFFER-LENGTH, BY REFERENCE BUFFER,
BY REFERENCE COMPLETION-CODE, REASON.

CALL 'MQGET' USING BY VALUE HCONN, BY VALUE Q-HANDLE,
BY REFERENCE MESSAGE-DESCRIPTOR,
BY REFERENCE OPTIONS,
BY VALUE BUFFER-LENGTH, BY REFERENCE BUFFER,
BY REFERENCE DATA-LENGTH, COMPLETION-CODE, REASON.

CALL 'MQCLOSE' USING by value HCONN, BY REFERENCE Q-HANDLE,

BY VALUE OPTIONS,
BY REFERENCE COMPLETION-CODE, REASON.

CALL 'MQDISC' USING BY REFERENCE HCONN, BY REFERENCE
COMPLETION-CODE, REASON.
```

AIX

Complete the following steps to configure the COBOL program on AIX:

1. Set the following environment variables, which are used by the application:

   `DEMO_WRITE_TO_MQ=Y`

   This turns on the WebSphere MQ functionality of the sample application. It causes a simple message to be written to a queue when a record is amended. If your system does not have WebSphere MQ installed, or you do not want to use it now, omit this variable, or set it to `N`.

2. Next, specify the queue manager using the following command:

   `DEMO_MQ_QUEUE_MANAGER=mqdemo`

   This specifies the name of the queue manager (as created previously with the **`crtmqm mqdemo`** command).

3. Specify the local queue using the following command:

   `DEMO_MQ_QUEUE=CICS.LOCAL.QUEUE`

   This specifies the name of the local queue that receives the message.

Windows

Complete the following steps to configure the COBOL program on Windows:

1. Set the following environment variables, which are used by the application:

   `DEMO_WRITE_TO_MQ=Y`

   This turns on the WebSphere MQ functionality of the sample application. It causes a simple message to be written to a queue when a record is amended. If your system does not have WebSphere MQ installed, or you do not want to use it now, omit this variable, or set it to `N`.

2. Next, specify the queue manager using the following command:

   `DEMO_MQ_QUEUE_MANAGER=mqdemo`

   This specifies the name of the queue manager (as created earlier with the **`crtmqm mqdemo`** command).

3. Specify the local queue using the following command:

   `DEMO_MQ_QUEUE=CICS.LOCAL.QUEUE`

Compilation (all OS)

When compiling programs to use with WebSphere MQ, use the compilation options **`-D5`** and **`-Dz`**. This option specifies that usage BINARY data items are treated as usage COMP-5, and that data items have their sizes calculated by their storage capacity, not their picture size.

3.5.6 WebSphere MQ and C

C programs access WebSphere MQ using the MQ*xxxx* routines available in the `/usr/mqm/lib/libmqic.a` (AIX) or `MQM.DLL` (Windows) library. Example 3-10 shows some example calls.

Example 3-10 Example calls

```
MQCONN(QMName, /*                    /* queue manager */
&Hcon,     /* connection handle */
&CompCode, /* completion code */
&CReason); /* reason code */

MQPUT(Hcon,                 /* connection handle */
Hobj,                        /* object handle */
&md,                        /* message descriptor */
&pmo,                       /* put message options */
(long int)sizeof(msg),      /* size of message data */
&msg,                      /* message data */
&CompCode,                 /* completion code */
&Reason);                  /* reason code */
```

3.5.7 WebSphere MQ triggers

In this section, we describe how to use the WebSphere MQ trigger monitor program supplied by CICS, and to trigger a CICS transaction to get the message from a WebSphere MQ queue.

WebSphere MQ trigger mechanism

Examine the mechanism of the WebSphere MQ queue manager (Figure 3-7).

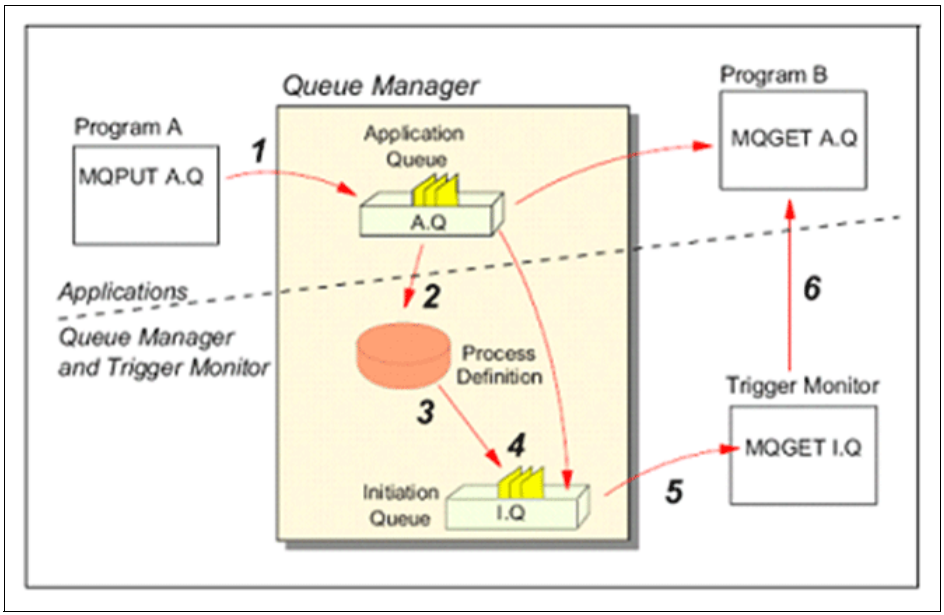


Figure 3-7 WebSphere MQ queue manager

A trigger for application queue `A.Q` is enabled. When the following conditions are met, the queue manager generates a trigger event:

- A trigger is enabled.
- A message arrives.
- The initiation queue attribute is defined.
- A process is defined.
- The process attribute of the application queue is defined.
- The application queue is not opening for output.
- The trigger type is matched.

With the trigger event, the queue manager generates a trigger message and puts it into the initiation queue defined by the application queue. The message content of the trigger message is an instance of the structure `MQTM` defined, as shown in Example 3-11.

Example 3-11 Message content of the trigger message

```
struct tagMQTM {
MQCHAR4 StrucId;  /* Structure identifier */
MQLONG Version; /* Structure version number */
MQCHAR48 QName; /* Name of triggered queue */
MQCHAR48 ProcessName; /* Name of process object */
MQCHAR64 TriggerData; /* Trigger data */
MQLONG ApplType; /* Application type */
MQCHAR256 ApplId;              /* Application identifier */
MQCHAR128 EnvData;           /* Environment data */
MQCHAR128 UserData;          /* User data */
};
```

The following elements are shown in Example 3-11:

- **`QName`** is the name of the triggered queue.
- **`Process`** is the name of the process object.
- **`TriggerData`** is the `TRIGDATA` attribute defined in the application queue.
- **`ApplType`** is the `APPLTYPE` attribute defined in the process definition.
- **`ApplId`** is the `APPLICID` attribute defined in the process definition.
- **`EnvData`** is the `ENVDATA` attribute defined in the process definition.
- **`UserData`** is the `USERDATA` attribute defined in the process definition.

The trigger monitor implements a loop to get messages from the initiation queue. When it gets a trigger message, the trigger monitor parses the message and starts the corresponding program in an asynchronous way, so that it can return immediately to wait for other trigger messages. The triggered application is started to process the message in the application queue.

The CICS MQ Adapter task initiator (CKTI) trigger monitor is included with TXSeries for monitoring WebSphere MQ trigger queues. For more information about how to use CKTI, see Chapter 4, “Enterprise integration” on page 117.

Writing a trigger application

A trigger application must be written to accept the parameter transfer mechanism of the trigger monitor. The parameters that the trigger application requires are the queue manager name and the queue name. The `accchgal.ccs` sample program, which is part of this book’s sample application, shows you a sample of a trigger application.

3.5.8 DB2

The sample application that we created demonstrates an XA connection to DB2, which it uses to add rows to a simple *audit trail* table. Whenever a customer record is updated, we add a row with a date and time, a record of the customer number that was affected, and a text description of the change.

The database table

This section describes how to create a database called `CICSTEST` on your DB2 instance:

1. Change user to the database's owner:

   ```
   su - db2local
   ```

2. Start the DB2 instance:

   ```
   db2start
   ```

3. Create the database:

   ```
   db2 create database CICSTEST
   ```

4. Run DB2 commands to create a simple table, as shown in Example 3-12.

   Example 3-12 Running DB2 commands

   ```
   db2
   CONNECT TO CICSTEST
   CREATE TABLE CUSTAUDIT (CA_DATE DATE, CA_TIME TIME, CA_CUSTOMER
   CHAR(9), CA_ACTION CHAR(30))
   QUIT
   ```

5. Add an entry to the XAD stanza to establish the XA connection to DB2, as shown in Example 3-13.

   Example 3-13 Establishing the XA connection to DB2

   ```
   cicsadd -c xad -r $CICSREGION XADB2 ResourceDescription="XA
   Definition
   to CICSTEST DB" SwitchLoadFile=cicsxadb2
   XAOpen="CICSTEST, db2local,
   db2local"
   ```

6. Use a COBOL program, called from the `CUSTUPDT` program, to add the row, using the code shown in Example 3-14.

Example 3-14 Code to add the row

```
EXEC SQL
INSERT INTO CUSTAUDIT VALUES
(:CA-DATE,
:CA-TIME,
:CA-CUSTOMER,
:CA-ACTION)
END-EXEC.
```

Remember: Because the DB2 connection is an XA one, no **`CONNECT`** and **`DISCONNECT`** logic is necessary in the COBOL program. The connection is made when the region starts.

7. To precompile the program, use the commands shown in Example 3-15.

Example 3-15 Precompiling the program

```
su db2local
db2 connect to cicstest
db2 prep CUSTSQL.sqb target ANSI_COBOL
quit
exit
export
COPYPATH=../Copybooks:/user/lpp/cics/include:${DB2DIR}/include/cobol_i
mv CUSTSQL.cbl CUSTSQL.ccp
cicstcl -lIBMCOB CUSTSQL.ccp
mv CUSTSQL.ibmcob
/var/cics_regions/$CICSREGION/bin
```

Information: Consider the following details about the code in Example 3-15:

- We chose to use **`su`** for the db2local user so that the **`prep`** command automatically performs an appropriate **`bind`** command. You can choose to do the **`bind`** command manually and omit the **`su`** step if you prefer. If you do, you must use a different **`connect`** command:

  ```
  db2 connect to cicstest user db2local
  ```

- Verify the path name of your DB2 `include` directory. It can differ from the one shown here.

8. Set the environment variable:

   ```
   DEMO_WRITE_TO_DB=Y
   ```

   This specifies that the application must write audit records to a DB2 table when account updates are done. If you do not have DB2 installed, or do not currently want to use this feature, omit the variable, or set it to **N**.

9. Use the following DB2 instance for the audit records:

   ```
   DB2INSTANCE=db2local
   ```

3.5.9 Oracle

This section demonstrates Oracle on Windows in the BigBlueBank application. The installation on AIX is similar:

1. Install Oracle, and use the Oracle SQL*Plus tool to create a table with the following command:

   ```
   CREATE TABLE CUSTAUDIT (CA_DATE CHAR(10), CA_TIME CHAR(8), CA_CUSTOMER
   CHAR(9), CA_ACTION CHAR(30));
   ```

2. The choice of data types is made purely to provide some level of similarity with the DB2 table used elsewhere in BigBlueBank, and remove any concerns over date and time formatting.

> **Important:** We log in as `scott` and use `tiger` as the password on our system. However, we advise you to choose a more secure login and password.

The switch-load file

To build the switch-load file, perform the following steps:

1. Start a command prompt from the Visual Studio .NET **Tools** menu under the Windows **Start** button.
2. Decide which switch-load file to build, by choosing the appropriate makefile:
   - Single-phase commit: `oracle1pc.mk`
   - Two-phase commit with dynamic registration: `oraclexa_dyn.mk`
   - Two-phase commit without dynamic registration: `oraclexa_sta.mk`
3. Change to the appropriate directory. We choose the two-phase commit with dynamic registration in BigBlueBank. Therefore, we change to the following directory:

   ```
   C:\opt\cics\examples\RM_support\Oracle
   ```

Requirement: The `oracle1pc.mk` makefile includes the following line:

```
$(ORACLE_HOME)\precomp\lib\orasql?.lib
```

You must change this for Oracle Release 10. Add a second question mark:

```
$(ORACLE_HOME)\precomp\lib\orasql??.lib
```

4. Set some environment variables required by the makefile, as shown in Example 3-16 (as always, check the path names for your particular system).

   Example 3-16 Setting environment variables required by the makefile

   ```
   set ORACLE_HOME=C:\oracle\ product_dir\db_1
   set CICSPATH=C:\opt\cics
   ```

5. Use the **`nmake`** command to build the switch-load file:

   ```
   nmake /f oraclexa_dyn.mk
   ```

6. Copy the output to the region's bin directory:

   ```
   copy oraclexa_dyn.dll c:\var\cics_regions\%CICSREGION%\bin
   ```

7. Add an entry to the XAD stanza to define the XA connection to Oracle, as shown in Example 3-17.

   Example 3-17 Defining the XA connection to Oracle

   ```
   cicsadd -c xad -r %CICSREGION% XAORA ResourceDescription="Oracle
   Definition to Oracle" SwitchLoadFile=oracle1pc.dll
   XAOpen="Oracle_XA+Acc=P/scott/tiger+SesTm=50+LogDir=C:\temp+DbgFl=1"
   ```

   This defines a connection using `scott` and `tiger` as the user and password, with a timeout for inactive transactions of 50 seconds. XA calls and returns are sent to a logging file in the `C:\TEMP` directory.

8. Add select permissions on the `dba_pending_transactions` table to the user that you specify in the XA open string (in our case, `scott`). Start a SQL*Plus session as a system administrator and issue the following command:

   ```
   grant select on dba_pending_transactions to scott;
   ```

9. Oracle installs its own *stub* XA handlers by default. You must change these so that an **`EXEC CICS SYNCPOINT`** commits the Oracle work. Run **`regedit`** for Windows registry editor and open the following key:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\KEY_OraClient10g_home1
   ```

   (This is the key name on our machine, yours can vary slightly.)

10. Right-click and select **New String Value** from the menu. You require the following string value:

    ```
    ORA_XA_REG_DLL
    ```

    The following value is the value for this example:

    ```
    %CICSPATH%\bin\libEncServer.dll
    ```

11. Use the Oracle Pro*COBOL precompiler to precompile the `CUSTSQL.sqb` program in the `Sources` directory of the demonstration application:

    ```
    procob CUSTSQL.sqb
    ```

12. Compile the resulting output file:

    ```
    rename CUSTSQL.cbl CUSTSQL.ccp
    cicstcl -lIBMCOB CUSTSQL.ccp
    ```

13. Copy the object file to the region's bin directory:

    ```
    copy CUSTSQL.ibmcob c:\var\cics_regions\%CICSREGION%\bin\
    ```

14. Add the environment settings in the region's environment file:

    ```
    ORACLE_HOME=C:\oracle\product_dir\db_1
    ORACLE_SID=Orcl
    DEMO_WRITE_TO_DB=Y
    ```

15. Restart the Windows machine and start the region.

3.5.10 CICS on z/OS

In this section, we establish an ISC connection to a CICS region running on z/OS. This system returns a Credit Score when passed a customer ID.

The `CREDAP.ccp` COBOL program source is supplied with the sample application:

1. A Virtual Storage Access Method (VSAM) file is required on the z/OS machine. Example 3-18 shows the specification for this file.

Example 3-18 Specification for the VSAM file

```
//CRATINGS EXEC PGM=IDCAMS,REGION=1M
//SYSPRINT DD SYSOUT=*
//AMSDUMP DD SYSOUT=*
//SYSIN DD *
DELETE CIWS.CREDIT.RATINGS.FILE PURGE CLUSTER
DEFINE CLUSTER -
(NAME(CIWS.CREDIT.RATINGS.FILE) -
VOLUMES(WORK1F) -
INDEXED          -
RECORDS(50)      -
```

```
KEYS(9 0)        -
RECORDSIZE(80 80) -
FREESPACE(5 5) -
SHAREOPTIONS(2))-
DATA                -
(NAME(CIWS.CREDIT.RATINGS.FILE.DATA)) -
INDEX               -
(NAME(CIWS.CREDIT.RATINGS.FILE.INDEX))
/*
```

2. The suitable keys to match the sample files on the TXSeries machines are shown in Example 3-19.

Example 3-19 Suitable keys to match the sample files on TXSeries

```
111111000
111222000
123456789
222222000
222333000
333333000
333444000
444444000
444555000
555555000
555666000
666666000
666777000
777777000
777888000
888888000
888999000
987654321
999999000
999999888
```

3. You can select appropriate credit ratings for each customer.

3.6 Object-oriented programming under TXSeries

TXSeries provides the flexibility of writing server programs in both modular and object-oriented programming languages. With object-oriented programming, programmers can perceive TXSeries services as *objects*. TXSeries supports object-oriented programming in Java and C++. With this support, you can accomplish the following tasks:

- Design and create new modern TXSeries server applications using an object-oriented architecture.
- Design and create reusable classes that can be used in and outside the TXSeries environment.
- Effectively use the strengths of various existing application classes and tools that are provided with the object-oriented language.
- Perceive TXSeries services as objects.

3.7 TXSeries server programming in Java

This section provides an introduction to Java programming in a TXSeries environment.

3.7.1 Introduction

In today's world, Java is perceived as a popular system-independent object-oriented language platform. The Java language is designed to be portable and architecture-neutral, and the byte code generated by compilation requires a machine-specific interpreter for execution.

TXSeries provides support to run JavaServer applications in a transactional environment. It provides a range of facilities with which we can access TXSeries services as objects in Java. Therefore, Java programming skills can be effectively used in writing modern server applications.

Using Java for writing TXSeries server applications does not require programmers to code or to know about **`EXEC CICS`** statements. Instead, they use *classes* or *objects* that are supplied by TXSeries to access a service. This method of accessing the services eliminates the need for translating the server Java program.

TXSeries provides various functional classes for programmers to perform relevant TXSeries functions. In the next section, we describe various functional classes.

3.7.2 TXSeries and Java virtual machine

As with any system, it is important to understand how TXSeries runs your Java program in a transactional environment. TXSeries uses the Java virtual machine (JVM) runtime support to run your JavaServer applications. This section describes the fundamentals of how TXSeries uses the Java run time to run your applications.

As illustrated in Figure 3-8 on page 85, TXSeries needs to initialize the JVM on the application server process that runs your JavaServer application. The JVM is loaded and initialized only if a request comes to an application server that requires a JavaServer program to be run.

Further requests to run a JavaServer program on the same application server process continue to use the JVM that was initialized earlier. If the request happens to go to another application server process, this loads and initializes a new JVM.

TXSeries supplies a language run time for Java called `cicsprJAVA`. This run time is responsible for loading and initializing the JVM and enabling JavaServer applications to run under the TXSeries environment. The `cicsprJAVA` run time is dynamically loaded onto the application server process upon a request to run a Java server program.

The run time is not unloaded after the server program execution completes. Instead, it is reused when another request comes to run a JavaServer program on the same application server process.

Figure 3-8 shows TXSeries initializing the JVM.

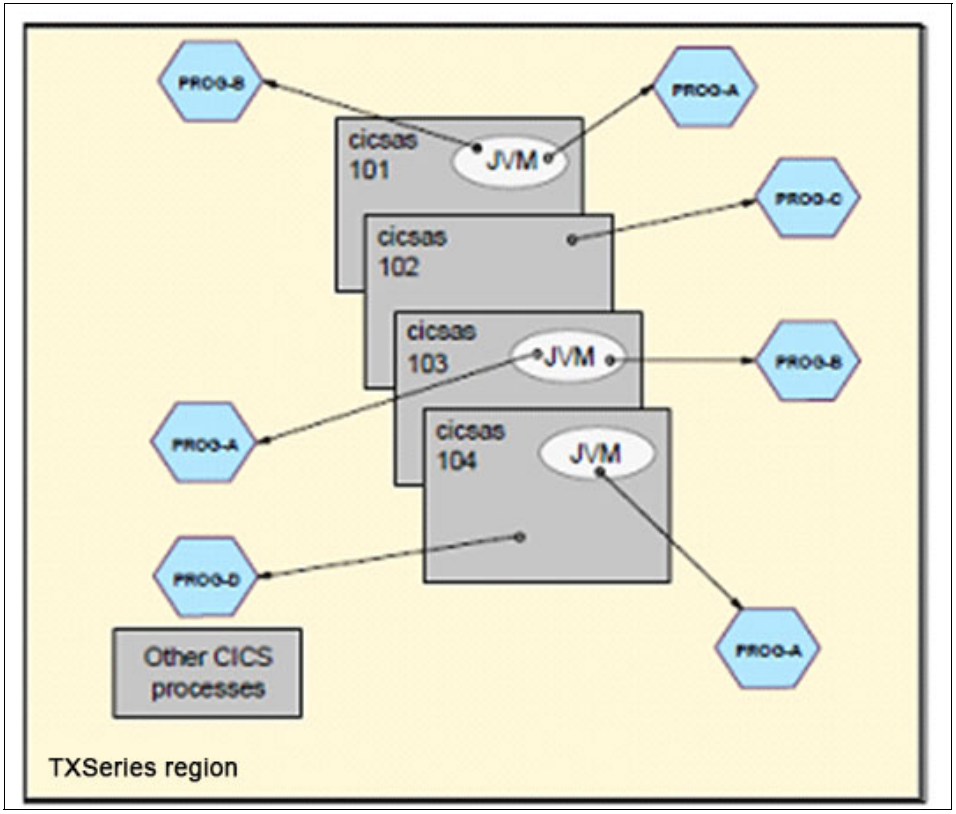


Figure 3-8 TXSeries CICS and JVM

Remember: The `cicsprJAVA` run time is not loaded on to the application server process unless a request to run a JavaServer program is received by that application server process.

In Figure 3-8, PROG-A and PROG-B are Java programs that require a JVM to be loaded within the CICS application server process. PROG-C and PROG-D are programs not based on Java.

3.7.3 Performance considerations

TXSeries does not cache JavaServer programs. This means that the `ProgramCache` program definitions attribute is not applicable for JavaServer programs. The caching takes place within the JVM, as JavaServer programs are directly interpreted by the Java run time.

Because there are multiple application server processes running, there can be multiple JVMs initialized and used within the region. Therefore, an application server process cannot share its JVM resources with another. When a request to run a JavaServer program is made, an application server process initializes the Java run time, loads the application classes, and runs them to completion.

When the same request is made again, and if this time the request is assigned to a different application server process, the new application server process must initialize a new JVM, load the application classes, and run them to completion. However, if the second request runs in the same application server as that of the first request, JVM is not loaded again, and therefore application classes might have been cached within the JVM.

Another factor that can affect the performance of the system is the use of the **`IdleTimeout`** region definitions (RD) parameter of the region. This is because as the **`IdleTimeout`** is reached for an application server process waiting for some work, the application server process is ended, so its corresponding JVM is also ended. This affects the performance because the JVM and the relevant application classes need to be reloaded onto a new application server process.

Tip: To avoid idle termination of the application server processes, we can either increase the **`IdleTimeout`** RD parameter to a higher value, or keep the `MinServer` and `MaxServer` values the same.

3.7.4 JCICS

Java class library for CICS (JCICS) is a fundamental programming element to write JavaServer applications in a CICS environment. TXSeries supplies the JCICS application programming interfaces (API), which are basically Java classes that can be instantiated in a Java program to access a given service. The APIs are broadly categorized to provide the following services:

Error handling and abnormal termination support

These services enable you to control the execution of the task, such as to cancel a task, issue a forced abnormal end of task (abend) to the class, handling conditions, and abends that are produced by TXSeries services. These services are in the *Task* Java class.

APPC mapped conversations
A complete Java class is provided to enable APPC-mapped conversations. These services are in the `AttachInitiator`, `Conversation`, and `ConversationPrincipalFacility` Java classes.

Terminal control
The services in this category are limited to only two APIs that are equivalent to **`SEND CONTROL`** and **`SEND TEXT`** CICS commands. These services are available in the Task Java class.

File control services
These services provide APIs to access VSAM files. The supported file types are key-sequenced data set (KSDS), entry-sequenced data set (ESDS), and relative record data set (RRDS). These services are available in `File`, `KeyedFile`, `KSDS`, `ESDS`, and `RRDS` Java class.

Timer services
These services provide the means to start or cancel tasks. Other time-related services, such as `ASKTIME`, `FORMATTIME`, and `DELAY`, are *not* supported. These services are available in the `StartRequest` Java class.

Program execution
These services enable you to call another program in the same logical unit of work (LUW). Data can be passed and received through COMMAREA. These services are in the `Program` class.

Logical unit of work services
These services enable you to commit or roll back the task's work. These services are in the `Task` class.

Serialization services
These services provide the means to handle shared access to resources. These services are in the `SynchronizationResource` class.

Temporary storage queue (TSQ) services
These services provide the means to share data across multiple transactions. These services are in the `TSQ` class.

Transient data queues (TDQ)
These services provide the ability to access TDQs. These services are in the `TDQ` class.

The following services are *not* supported by JCICS under TXSeries:

- DUMP services
- JOURNAL services
- Storage services
- BMS services
- APPC unmapped conversations
- Authentication services
- Inquire services for Programs, Files, Journals, and Statistics

In CICS, there are two primitive structures that are used: COMMAREA and EXEC interface block (EIB). The COMMAREA is automatically passed into a program by using the `CommAreaHolder` argument to the main method, and there is no class that can be instantiated to access the EIB block. However, certain fields, such as Transaction ID (`eibtrnid`) and Task Number (`eibtaskn`) can be accessed through **`Task.getTransactionName()`** and **`Task.getTaskNumber()`**.

3.7.5 Exception handling

It is always necessary to handle error conditions in your code, because anything can go wrong within the application. The error reporting and handling in JCICS is integrated into the standard Java exception handling mechanism.

In other languages, TXSeries indicates the success or failure of a CICS command through EIB variables called `EIBRESP` and `EIBRESP2`. In Java, these `EIBRESP` codes are mapped to Java exception classes. For each `EIBRESP` value that can occur in CICS, there is one corresponding Java exception class.

The exceptions are generally classified as *checked exceptions* and *unchecked exceptions*. When you call a method that might produce a checked exception, you are requested to either handle the exception in your method, or to declare your own method as producing that exception. However, unchecked exceptions need not necessarily be handled or declared. They usually represent conditions that are so common (or so rare) that it is impractical to be forced to handle them.

3.7.6 Components involved

The following components are involved when writing a Java program under the TXSeries environment:

cicsprJAVA The Java language runtime environment supplied by TXSeries.

dfjcics.jar The JCICS API class library.

com.ibm.cics.server The JCICS API package name. This is the package that you must import in your Java application program.

Java runtime environment (**JRE**)
Java runtime necessary to run a Java program.

3.7.7 JCICS class hierarchy

Figure 3-9 illustrates the JCICS class hierarchy.

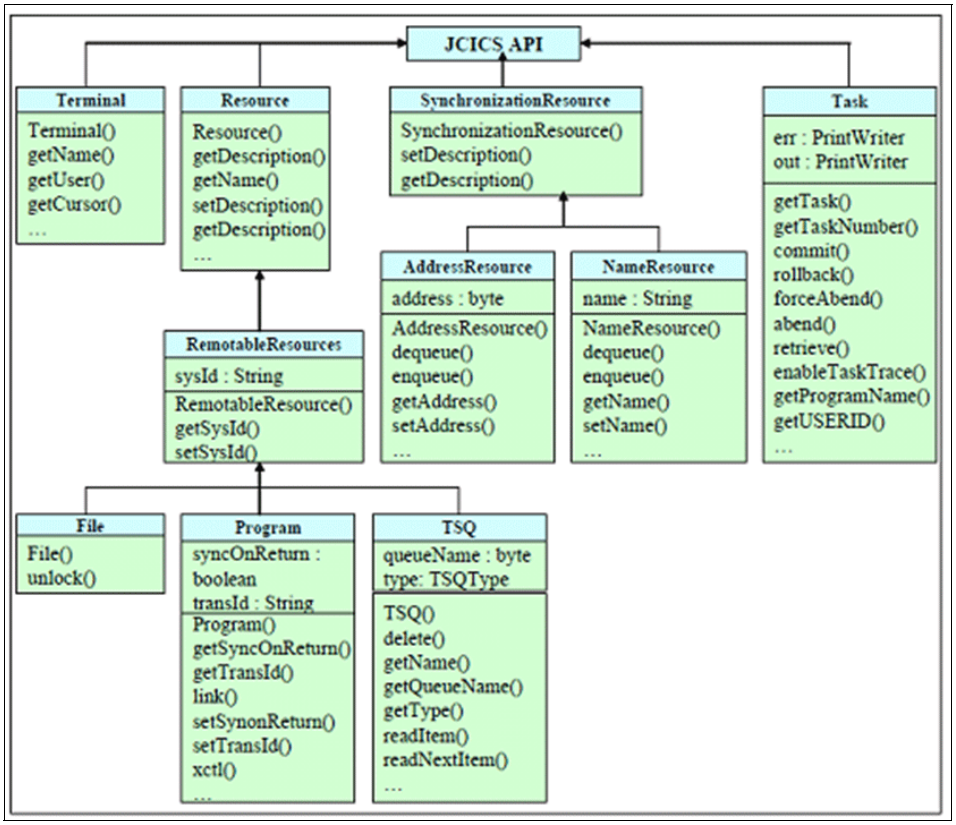


Figure 3-9 JCICS API

3.7.8 Setting up CICS for running Java under TXSeries

In this section, we describe the steps of developing and deploying a Java application program in a TXSeries environment. For demonstration purposes, we use the BigBlueBank system's Get Credit Rating (BANKCRED) application. The BANKCRED application accepts a customer ID known to the bank, and returns the corresponding credit rating or score of that customer. The credit rating of each customer is on a mainframe system running CICS.

We implement the following classes for this application:

BankCredApp.java The main application class, which receives a COMMAREA containing the customer ID, and returns the corresponding credit rating, if any.

DplLink.java The `DplLink` class is used by BankCredApp to perform a distributed program link (DPL) to a remote CICS program.

LogMsg.java The `LogMsg` class is used by all classes in this project to log informational, warning, or error messages. The `LogMsg` class places messages on to a TSQ called `JTSQ`.

VsamKsds.java The `VsamKsds` class is used by BankCredApp to browse through a VSAM KSDS file. The `VsamKsds` class implements an Iterator class to browse.

The sources for these files are provided with this book. We show you how to develop the JCICS application using the Rational Application Developer software environment.

Developing the BANKCRED application using Rational Application Developer

We use Rational Application Developer on Windows to develop a Java application, and deploy it to the TXSeries environment. As a prerequisite, the following software must be installed or configured on your machine:

- Rational Application Developer V6.0
- Java Development Kit (JDK)/JRE 1.4 or later
- TXSeries V7.1 or later

We perform the following steps:

1. Develop the BANKCRED application using Rational Application Developer.
2. Deploy the BANKCRED application to the TXSeries environment.
3. Define the BANKCRED application to the TXSeries region.
4. Run the BANKCRED application.

Developing the BANKCRED application using Rational Application Developer

This section demonstrates how to create the BANKCRED application using the Rational Application Developer tool:

1. Start the IBM Rational Developer, as shown in Figure 3-10.

Figure 3-10 Starting the Rational Application Developer tool

2. Switch to the Java perspective by selecting **Window** → **Open Perspective** → **Java**. This is shown in Figure 3-11.

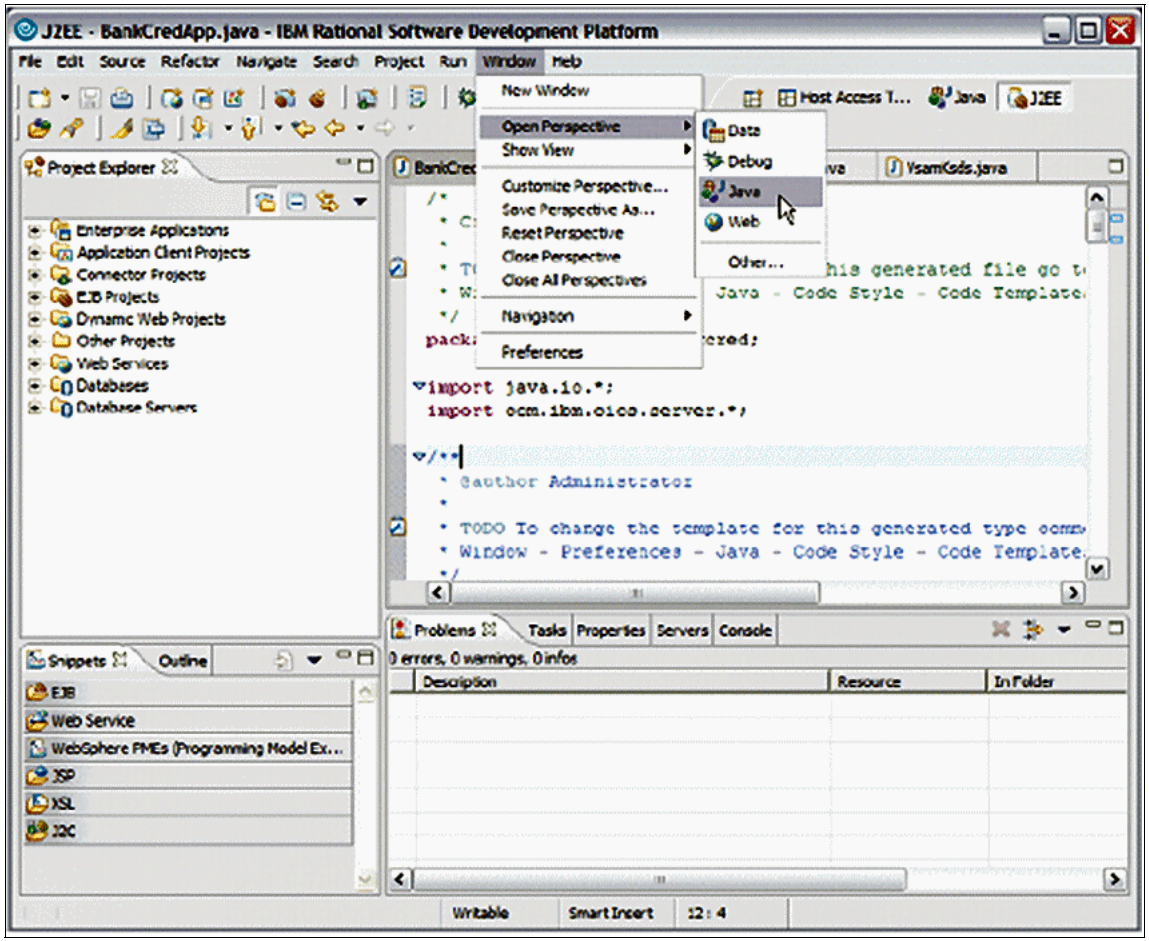


Figure 3-11 Java perspective

3. Create a new Java project by selecting **File** → **New** → **Project**.
4. In the New Project window, select **Java Project** and click **Next**.

5. In the New Java Project window, enter the Project name `BANKCRED` and click **Finish**, as shown in Figure 3-12.

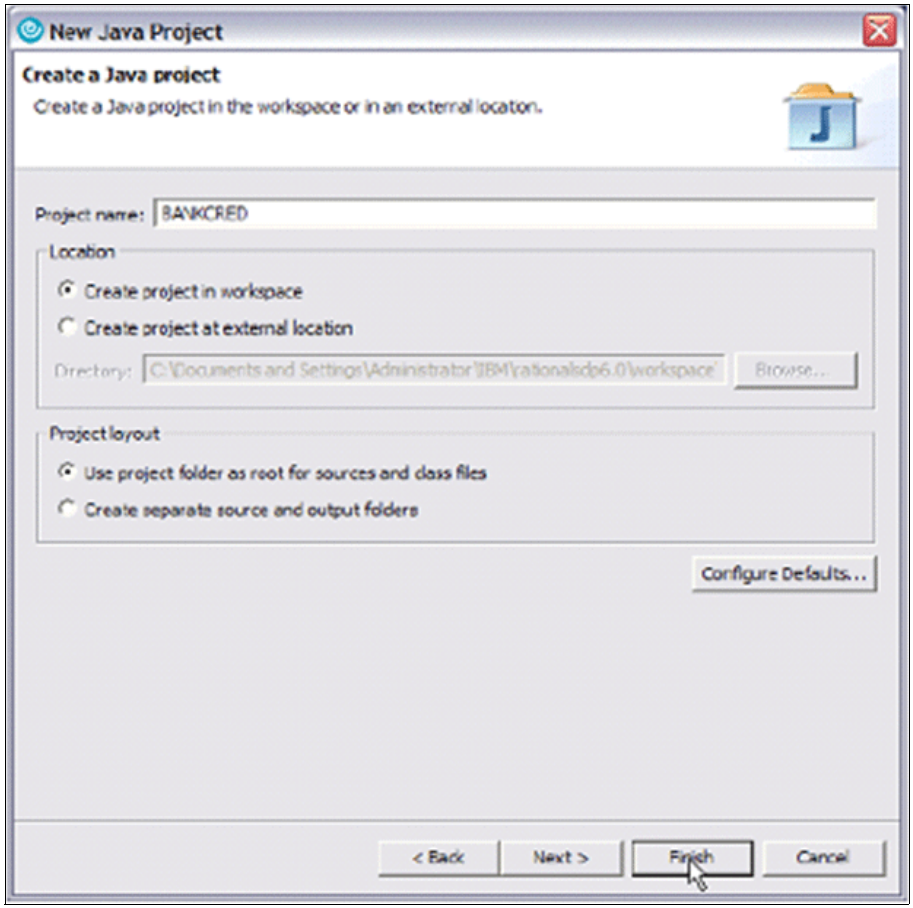


Figure 3-12 Creating a new Java project

6. Create a Java package. Select **New** → **Package**.
7. In the new window, enter `com.ibm.bank.bankcred` as the package name and click **Finish**.
8. Create a new Java class called BankCred under the `com.ibm.bank.credapp` package. Select **File** → **New** → **Class**.

9. In the New Java Class window, enter `BankCredApp` and click **Finish**, as shown in Figure 3-13.

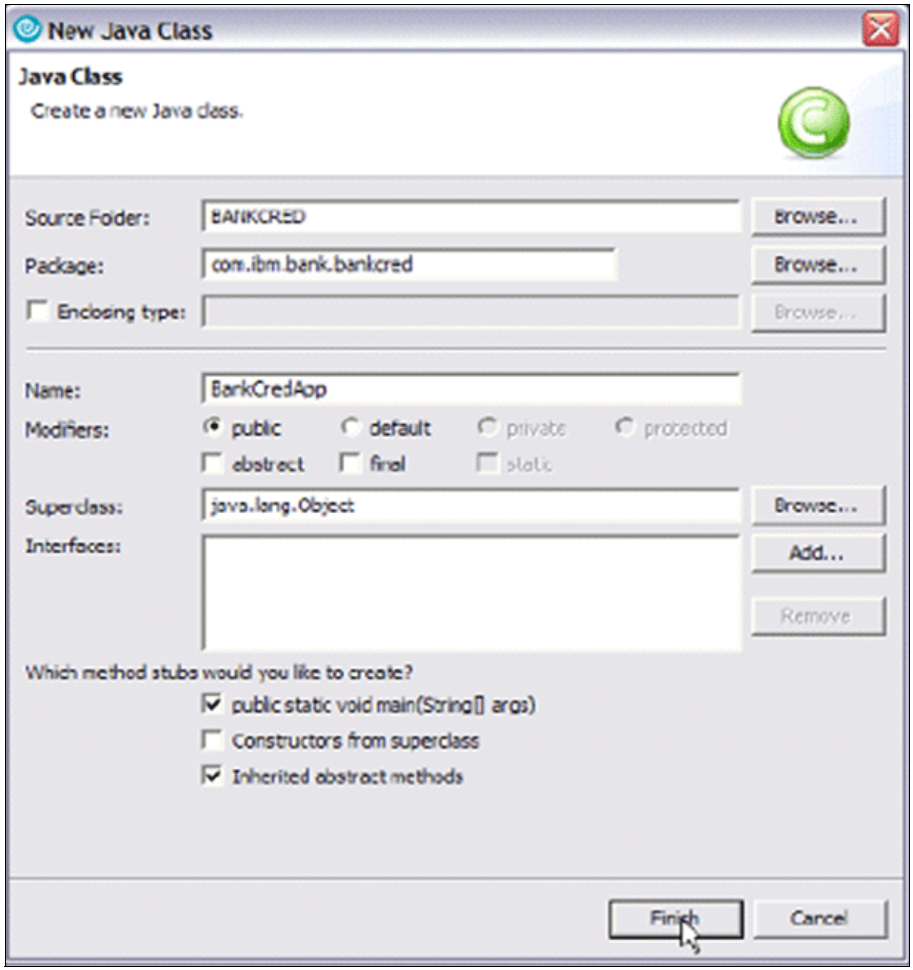


Figure 3-13 Creating a new Java class

10. Also create new classes for the `DplLink`, `LogMsg`, and `VsamKsds` interfaces.
11. Add the CICS JAR file (`dfjcics.jar`) that contains the CICS API classes to the project build path. To do this, select the **BANKCRED** project, and press Alt+Enter key.
12. The properties window for the BANKCRED project opens:
    a. Select **Java Build Path** from the left pane of the window.
    b. Switch to the **Libraries** tab.
    c. Click **Add External JARs**.

13. This opens another window called JAR Selection. Browse to the location of `dfjcics.jar`. Select the `dfjcics.jar` file and click **Open**.

    Figure 3-14 shows the properties of the project after adding the CICS JAR file.

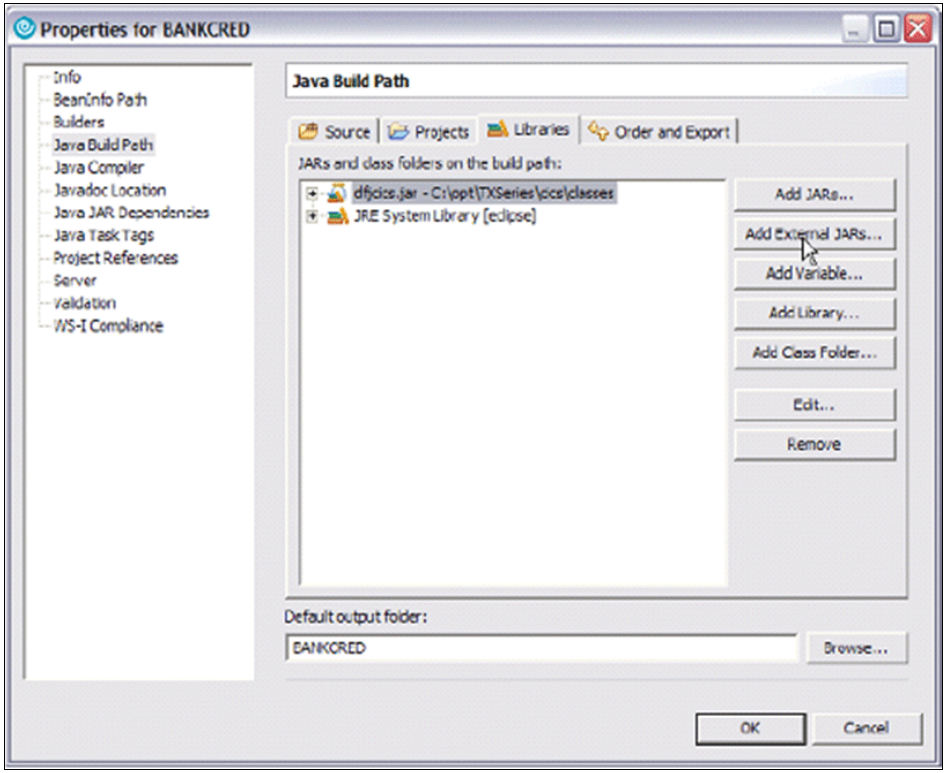


Figure 3-14 Properties of the project

14. This completes the setup of the Rational Application Developer environment for developing JCICS applications. You can add new classes, and access JCICS services, by importing the following line in your new class:

```
import com.ibm.cics.server.* ;
```

15. Figure 3-15 shows the main Rational window.

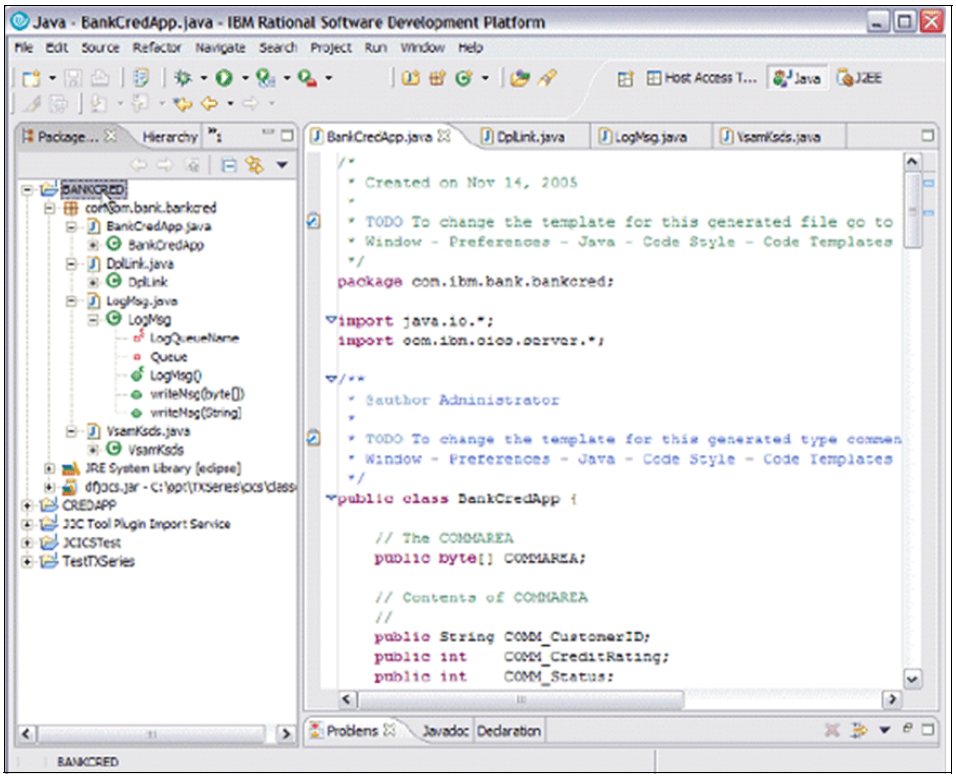


Figure 3-15 Main Rational window

Deploying the application from Rational Application Developer to the TXSeries region

You can now deploy the developed JCICS application to the TXSeries region. You have to create a region or use an existing region to deploy the JCICS classes from Rational Application Developer:

1. In the region's environment file, ensure that you have the following variables that are required to run Java applications:
   - `JAVA_HOME=<location of JDK/JRE 1.4.2>`
   - `CLASSPATH=/usr/lpp/cics/classes/dfjcics.jar:/var/cics_regions/javareg/classes:<other classpaths>`
2. To deploy the JCICS application developed from Rational Application Developer, right-click the project name **BANKCRED** and select **Export**.

3. An Export window opens, as shown in Figure 3-16. Choose File Transfer Protocol (**FTP**) if your region is running on another system, such as AIX. Otherwise, use the File system if the region is running on the local machine. In this example, we are running the BigBlueBank system on another machine. Therefore, we select **FTP** and click **Next**.

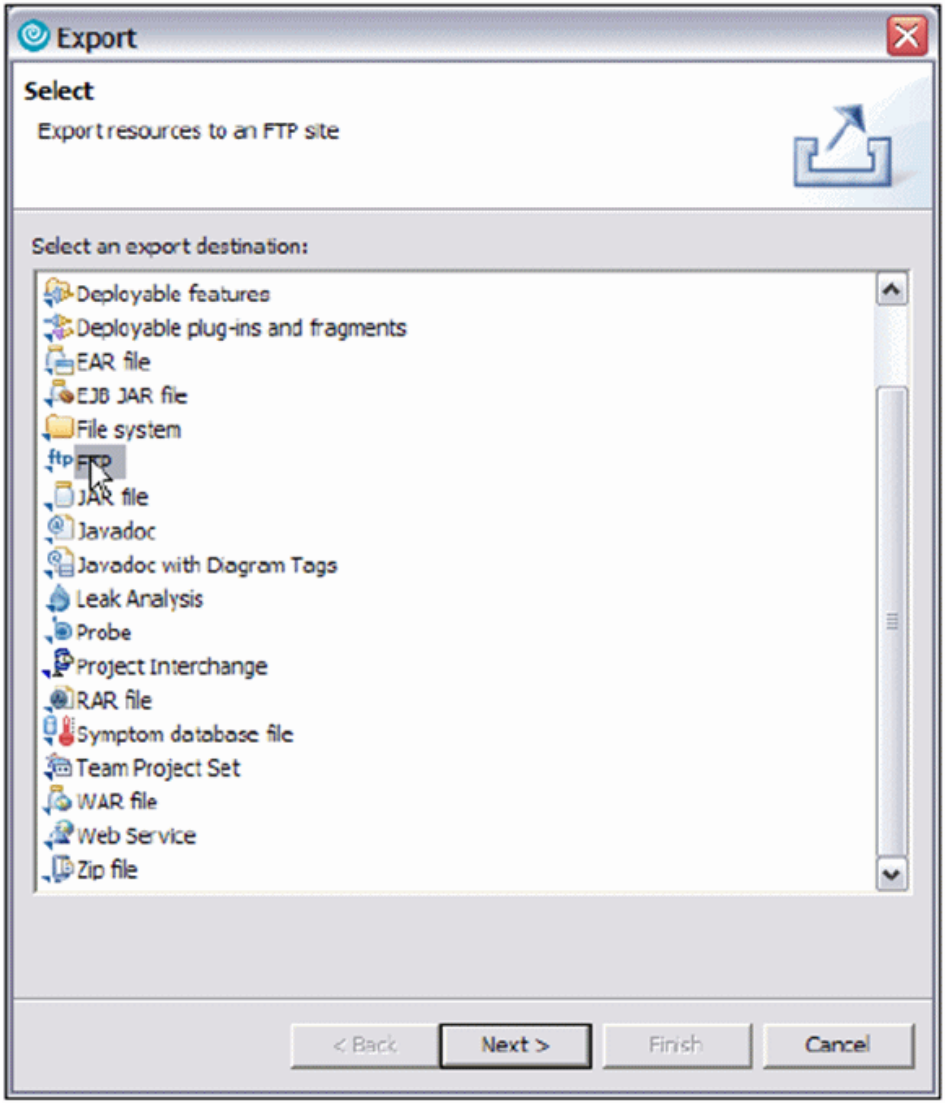


Figure 3-16 Export window

4. In the next window, enter the remote host name in the FTP host field and enter the region's classes directory in the FTP folder field, as shown in Figure 3-17. Click **Next** to enter the remote machine's login user ID and password. Click **Finish** to deploy.

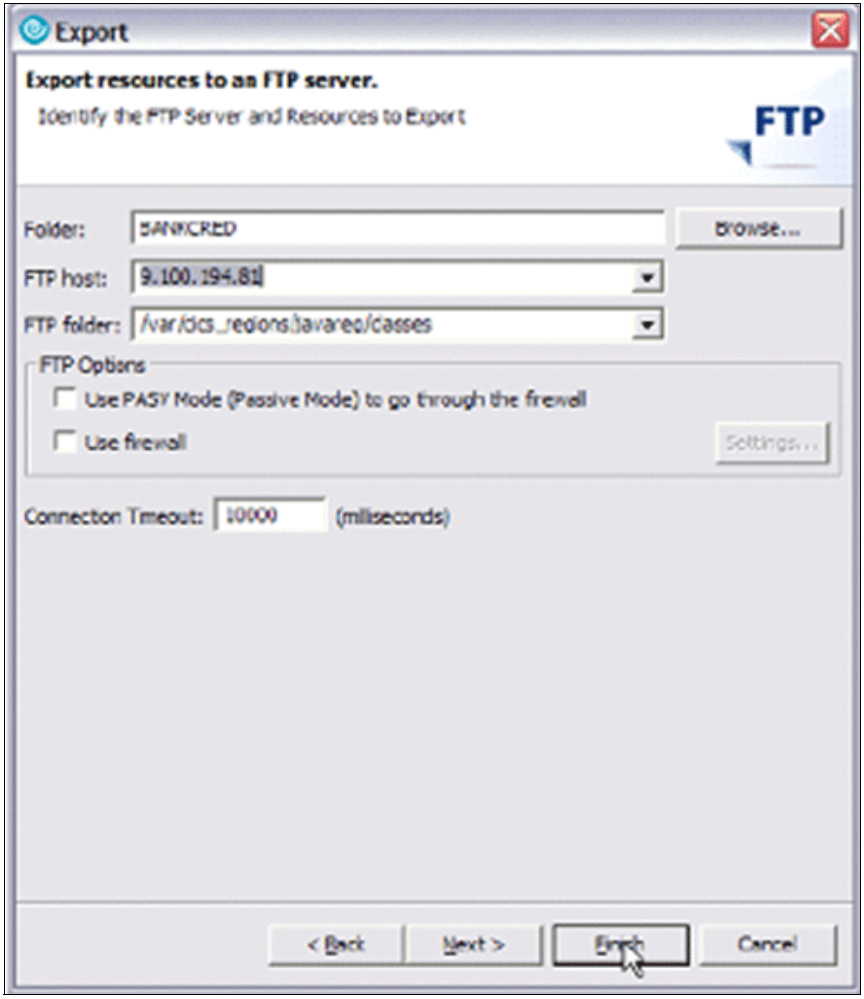


Figure 3-17 Export window after choosing FTP

5. The FTP Export Statistics window opens, as shown in Figure 3-18.

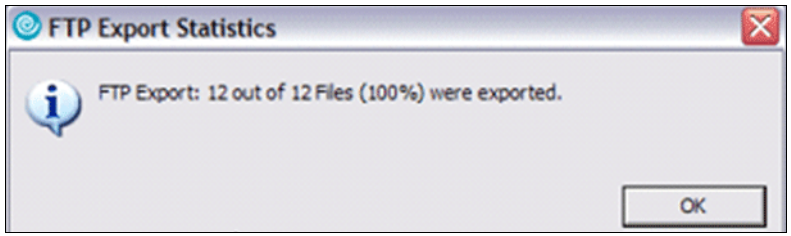


Figure 3-18 FTP Export Statistics window

You have completed the deployment of the JCICS application developed under Rational Application Developer to an AIX region running the BigBlueBank system.

6. Log on to the AIX machine to verify whether the deployed files are in the region's classes directory. Example 3-20 shows the files that were deployed from Rational Application Developer.

Example 3-20 Deployed files

```
{pons2:root}/var/cics_regions/javareg/classes -> ls -l
total 16

-rw-r----- 1 root system 305 Nov 22 17:10 .classpath
-rw-r----- 1 root system 507 Nov 22 17:10 .project
drwxr-x--- 3 root system 256 Nov 22 17:08 com
{pons2:root}/var/cics_regions/javareg/classes -> pwd
/var/cics_regions/javareg/classes
{pons2:root}/var/cics_regions/javareg/classes -> ls
com/ibm/bank/bankcred/
BankCredApp$BANKCUST.class LogMsg.class
BankCredApp.class LogMsg.java
BankCredApp.java VsamKsds$VsamIterator.class
DplLink.class VsamKsds.class
DplLink.java VsamKsds.java
{pons2:root}/var/cics_regions/javareg/classes ->
```

7. You have to add the program definition for the JCICS application into the region. Example 3-21 shows the command to do this.

Example 3-21 Command to add JCICS application to the region

```
{pons2:root}/ -> cicsadd -c pd -r javareg BANKCRED \
> PathName="com.ibm.bank.bankcred.BankCredApp.class" RSLKey=public
{pons2:root}/ -> cicsget -c pd -r javareg BANKCRED

BANKCRED:
GroupName=""
ActivateOnStartup=yes
ResourceDescription="Program Definition"
AmendCounter=0
Permanent=no
EnableStatus=enabled
RemoteSysId=""
RemoteName=""
TransId=""
RSLKey=public
PathName="com.ibm.bank.bankcred.BankCredApp.class"
ProgType=program
UserExitNumber=0
TemplateDefined=no
Resident=no
{pons2:root}/ ->
```

8. You have to restart the CICS region for the new program definition to take effect, because you have added the definition to the permanent database. When the region is restarted, the BigBlueBank system is ready to use the new JCICS application (BANKCRED) to retrieve the credit rating for a given customer.

Developing the BANKCRED application using cicstcl

If you do not have the Rational Application Developer tool, you can still develop JCICS applications and run them under the TXSeries environment. To do this, you have to write the JCICS application using an editor, and compile them using the javac compiler. We show you how to prepare the BANKCRED application without using the Rational Application Developer tool. To do so, complete the following steps:

1. Use an editor or other Java development tool to write the JCICS application.
2. Place the Java source files in the region's classes directory.

3. Set the environment variables:
   - `JAVA_HOME=/usr/java14`
   - `CLASSPATH=/usr/lpp/cics/classes:/usr/lpp/cics/classes/dfjcics.jar:/var/cics_regions/javareg/classes:$CLASSPATH`
4. Under the region's classes directory, run the following command:

   ```
   javac -d . BankCredApp.java DplLink.java LogMsg.java VsamKsds.java
   ```

5. When the compilation is successful, you find the class file under the `com` directory of the region's classes directory, as shown in Example 3-22.

   Example 3-22 Compiling the Java applications from command line

   ```
   {pons2:root}/var/cics_regions/javareg/classes -> export
   CLASSPATH=/usr/lpp/cics/classes:\
   > /usr/lpp/cics/classes/dfjcics.jar:\
   > /var/cics_regions/javareg/classes:$CLASSPATH
   {pons2:root}/var/cics_regions/javareg/classes
   ->{pons2:root}/var/cics_regions/javareg/classes -> ls
   BankCredApp.java DplLink.java LogMsg.java VsamKsds.java
   {pons2:root}/var/cics_regions/javareg/classes ->
   {pons2:root}/var/cics_regions/javareg/classes -> javac -d . *.java
   {pons2:root}/var/cics_regions/javareg/classes ->
   {pons2:root}/var/cics_regions/javareg/classes -> ls
   BankCredApp.java DplLink.java LogMsg.java VsamKsds.java
   com
   {pons2:root}/var/cics_regions/javareg/classes ->
   {pons2:root}/var/cics_regions/javareg/classes -> ls
   com/ibm/bank/bankcred/
   BankCredApp$BANKCUST.class DplLink.class
   VsamKsds$VsamIterator.class
   BankCredApp.class LogMsg.class
   VsamKsds.class
   {pons2:root}/var/cics_regions/javareg/classes ->
   ```

6. Define program definitions for the BankCredApp class and restart the region.
7. Test the BankCredApp JCICS application with the BigBlueBank system.

3.7.9 Problem determination

We describe problem determination techniques in detail in Chapter 6, "Troubleshooting" on page 329.

3.7.10 Configuring JVM under TXSeries

TXSeries enables you to pass more options to the JVM, such as `-verbose`, and `-verbose:gc`, during its initialization. This is done by passing the JVM options through the **`CICS_JAVA_OPTIONS`** environment variable. You can define this environment variable in the region's environment file.

3.7.11 Debugging

TXSeries provides a supplied transaction called CJDB that is used to debug Java applications. The CJDB transaction configures the IBM distributed debugger to debug Java applications running under the TXSeries environment.

3.7.12 Hints and tips

This section provides some hints and tips:

- Avoid the use of the **`System.exit()`** call in the Java program, because this ends the application server process abruptly. CICS initializes the environment for the Java applications, but more importantly, it performs cleanup for any processes that are used during the life of the application. Exiting from the application does not allow this cleanup process to run, and might lead to data inconsistency.
- Use **`CICS_JAVA_OPTIONS`** accordingly to pass JVM options to fine-tune the JVM according to your environment.
- Sometimes, you might find messages from the JVM in the TXSeries logs. These messages and other debugging messages that are written to `stderr` or `stdout` (for example, the output from the JVM when switches, such as `-verbose`, are used) are redirected to the region's `console.`*`nnnnnn`* and `console.msg`. This helps to locate information during problem determination.
- You can use `System.out` and `System.err` streams to print output as you do with any other stand-alone Java application. Alternatively, if the transaction is started from a terminal, you can use the `out` and `err` fields of the task class that direct the output to the user's terminal. If the task is *not* started through a terminal, the task class `out` field corresponds to the `System.out`, and the task class `err` field corresponds to the `System.err`.

3.8 CICS server programming in C++

This section provides an introduction to C++ programming in TXSeries.

3.8.1 Introduction

TXSeries supports writing CICS server applications in the C++ programming language. You can reuse existing application class libraries or templates, and the rapid cross-platform development tools that are available to help write CICS server applications. With C++, you can create reusable components quite easily, and use them in a CICS system, or a system not based on CICS (for example, batch applications).

3.8.2 IBM CICS Foundation Classes

The CICS Foundation Classes are an API for writing CICS server programs in the C++ programming language. They are well-designed class libraries that make C++ and CICS development much easier.

3.8.3 Components involved

The following components are involved when writing CICS server programs using C++ programming language:

cicsprCpp
This is a C++ programming language run time supplied by TXSeries. The run time is loaded on to the application server process before running any CICS server program that is compiled with C++ compiler.

C++ compiler
TXSeries supports Microsoft C++ compiler on Windows, and for IBM VisualAge® C++ compiler on AIX.

3.8.4 CICS Foundation Class library

All CICS Foundation Classes are derived, directly or indirectly, from the `IccBase` base class. These classes enable common interfaces to be defined for categories of classes. You can also use them to create your own derived classes. Extending the functionality provided by the CICS Foundation Classes is much simpler through an object-oriented approach. In general, the following class categories are defined:

Base and derived classes
These are the fundamental classes that are derived from the `IccBase` base class, such as `IccException` (for exception handling) and `IccBuf` (for manipulating buffers).

Resource identification classes

These classes model the behavior of major CICS resources, such as `IccTerminal` (for Terminal), `IccProgram` (for Programs), and `IccTempStore` (for temporary storage queues).

Resource classes

These classes help to define CICS resource identifiers, such as file definitions (FD), program definitions (PD), and transaction definitions (TD) stanza entries. These classes improve type checking. For example, methods that expect an `IccField` object as a parameter do not accept an `IccProgramId` object instead.

Record index classes

These classes are used to identify records in CICS files, such as `IccKey` (for KSDS file) and `IccRRN` (for RRDS file). They specify the record access method and identify which record in a file is to be retrieved, modified, or deleted.

Figure 3-19 illustrates some of the classes that you can find in CICS Foundation Classes.

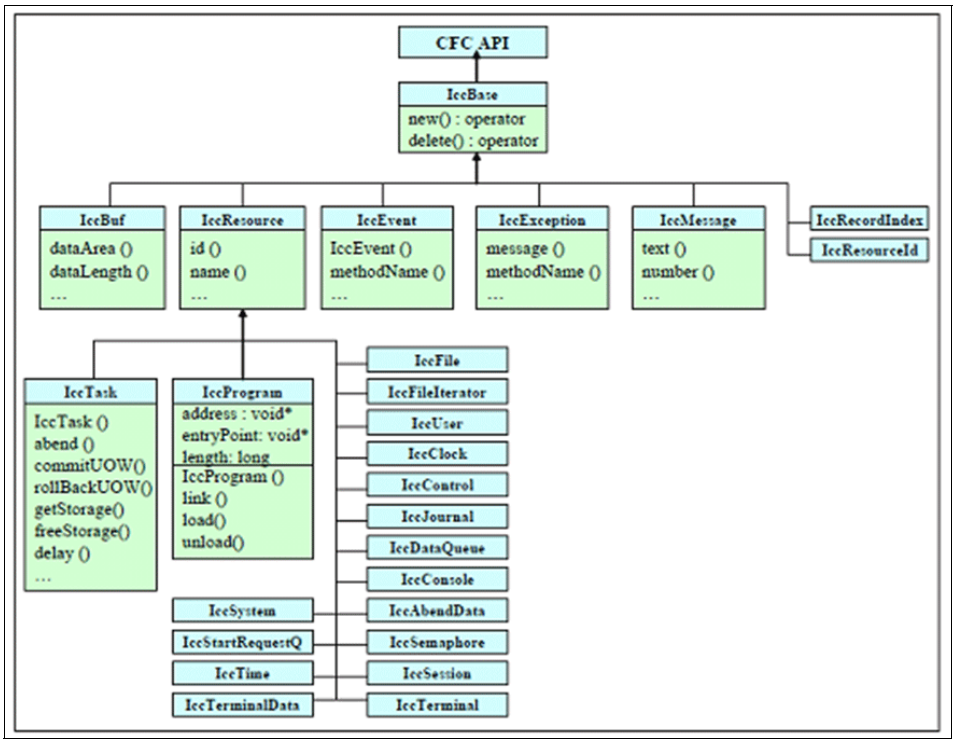


Figure 3-19 CICS Foundation Classes API

3.8.5 Running CICS Foundation Classes under TXSeries

This section describes the steps of developing and deploying a C++ application program in a TXSeries environment. In general, no additional configuration is required to set up CICS for running C++ applications.

The C++ programming language run time (`cicsprCpp`) is prebuilt, and is supplied with the TXSeries product. If your program has any dependency on third-party libraries or modules, you might have to update environment variables, such as **PATH** in the region's environment file or the system environment, which are inherited when the region is started.

Along with the C++ programming language run time, TXSeries supplies CICS Foundation Class header files and CICS Foundation Class dynamic link libraries (DLLs) that are required when building C++ applications that use CICS Foundation Classes.

> **Remember:** The CICS Foundation Classes do *not* require the CICS translator to be called.

Developing C++ applications using Microsoft Visual Studio

We use Visual Studio on Windows to develop a C++ application, and deploy it to the TXSeries environment. As a prerequisite, you require the following software installed or configured on your machine:

- Visual Studio
- TXSeries V7.1 or later

We perform the following steps:

1. Develop the BigBlueBank core applications using Visual Studio.
2. Develop the BANKCRED application using Visual Studio.
3. Deploy the BANKCRED application to the TXSeries environment.
4. Define the BANKCRED application to the TXSeries region.
5. Run the BANKCRED application.

Developing the BigBlueBank core applications using Visual Studio

For demonstration purposes, we identified the following BigBlueBank applications to be developed under Visual Studio:

CustBrowse.ccs	The business logic to browse through the existing customer records.
CustECI.ccs	The CORE business logic handling customer requests.
CustGet.ccs	Gets the customer accounts corresponding to a customer ID.
CustSignOn.ccs	Logs the customer into the BigBlueBank system.
CustUpdate.ccs	Updates the customer accounts corresponding to a customer ID.
CustUtility.c	Miscellaneous program for conversions and validations.

These applications are COMMAREA-based programs, and do not implement any presentation interface to the BigBlueBank system.

To create the BigBlueBank core applications using Visual Studio, complete the following steps:

1. Start Visual Studio 2010 or Higher.
2. Open the project or solution by selecting **File** → **Open** → **Project/Solution**.

3. In the Open Project window (Figure 3-20), perform the following tasks:
   a. Navigate to the BigBlueBank Folder in the `MSVC.NET Project` folder.
   b. Click **BigBlueBank.sln** to open the solution.
   c. Click **Open**.

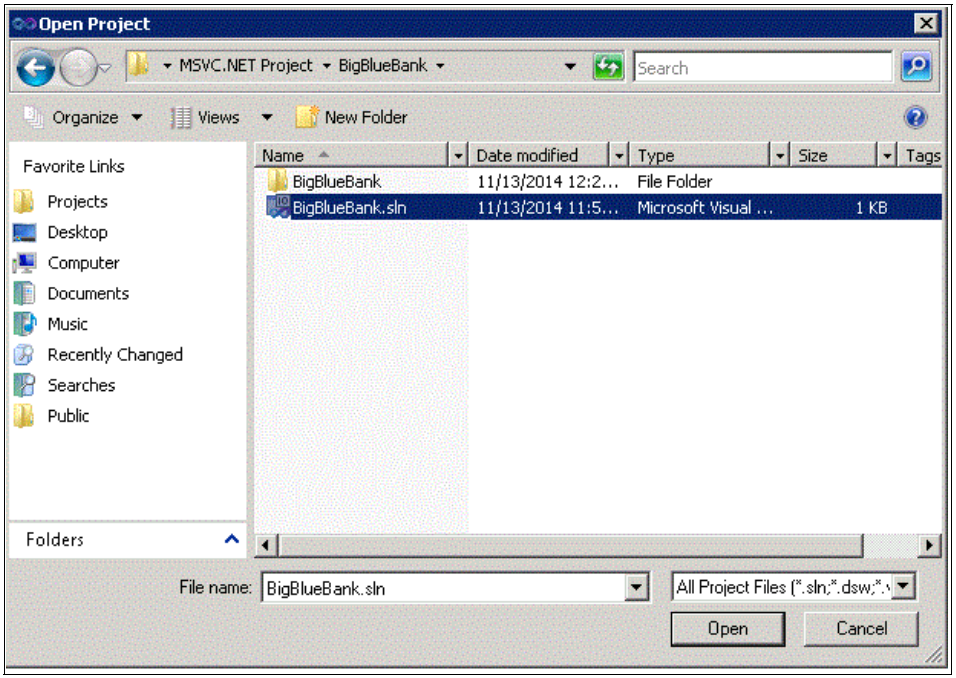


Figure 3-20 Open project window

4. The Solution Explorer window opens the BigBlueBank project, as shown in Figure 3-21.

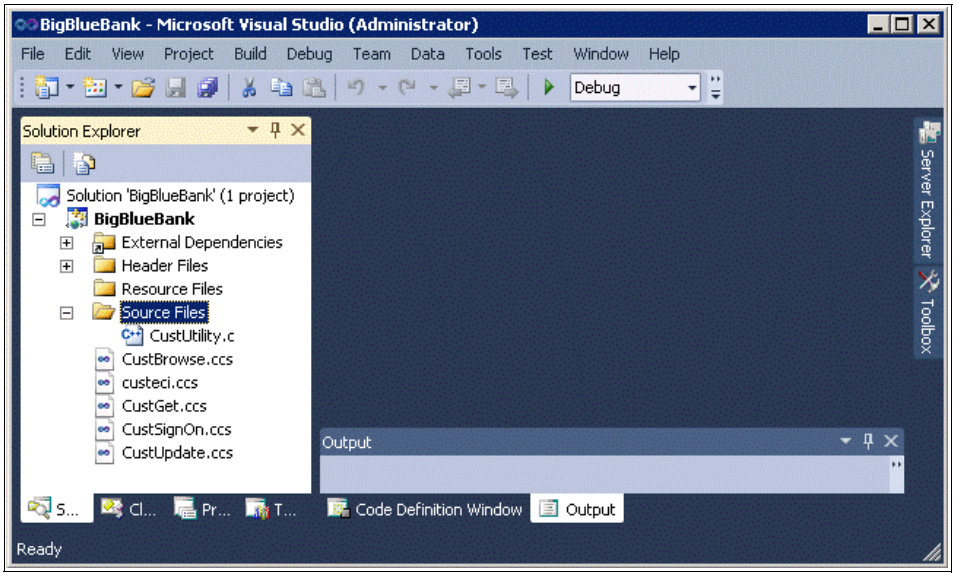


Figure 3-21 Solution Explorer window

5. All the C source files that are provided with this book will be loaded to the BigBlueBank project automatically.

6. Figure 3-22 shows the window after importing all of the source files to the project.

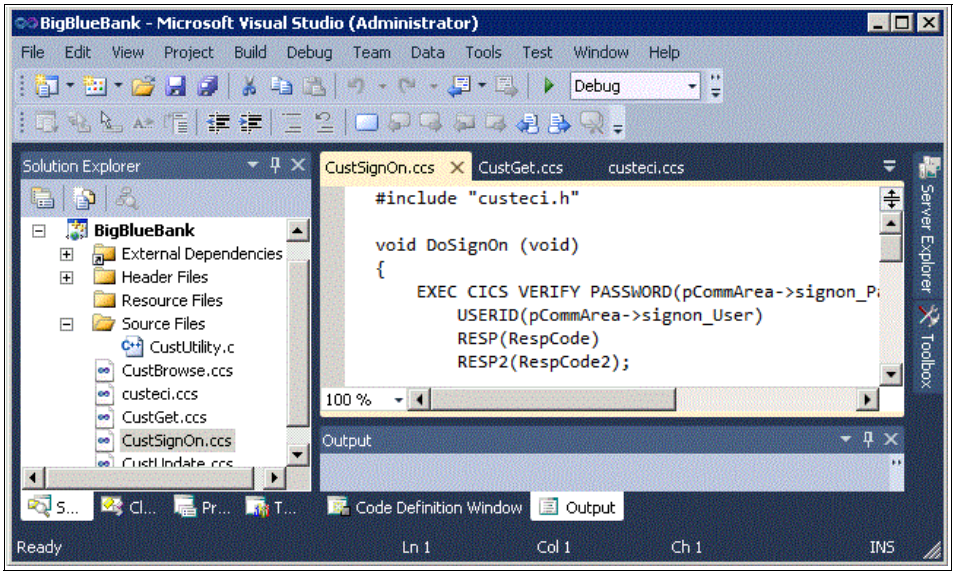


Figure 3-22 Source files after import

7. If you build the solution now you will get many errors. This is because Visual Studio does not recognize how to build the `.ccs` CICS source files. You have to set up custom builds for each of these `.ccs` source files. Set the include path containing the TXSeries application development header files, in Microsoft Visual Studio. To do so, select **Project** → **Properties**.

8. In the BigBlueBank Property Pages window (Figure 3-23), perform the following steps:
   a. Select **Configuration Properties** → **VC++ Directories**.
   b. On the right pane, select **Include Directories** and click the down arrow on the end of the line, and then click **edit.**
   c. Select the include path where the TXSeries application development header files are located, which is `C:\opt\cics\include`.
   d. In the Include Directories window, click the **NewLine** icon and type `C:\opt\cics\include`. Click **OK** to finish. Click **OK** to close the BigBlueBank Property Pages window.

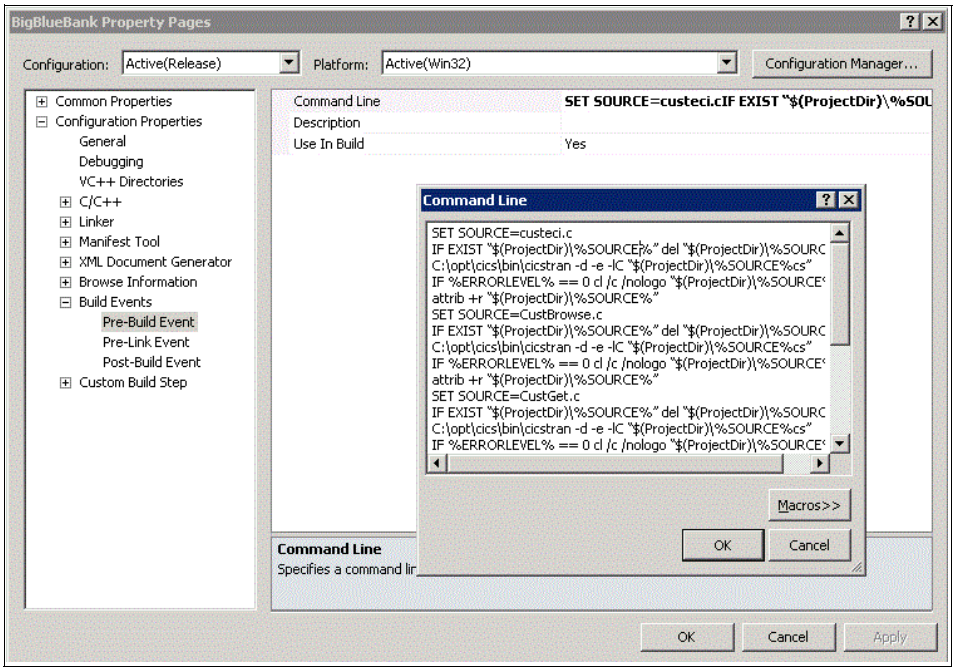


Figure 3-23 Selecting the source files

Now build custom build step rules for compiling the `.ccs` CICS source files.

9. Right-click BigBlueBank solution and select **Properties**.

10. The property page for the custom build window opens, as shown in Figure 3-24:

    a. Expand **Build Events** and select the sub option **Pre-Build Event**.

    b. On the right pane, enter the Command Line field:

    ```
    SET SOURCE=custeci.c
    IF EXIST "$(ProjectDir)\%SOURCE%" del "$(ProjectDir)\%SOURCE%"
    C:\opt\cics\bin\cicstran -d -e -lC "$(ProjectDir)\%SOURCE%cs"
    IF %ERRORLEVEL% == 0 cl /c /nologo "$(ProjectDir)\%SOURCE%"
    attrib +r "$(ProjectDir)\%SOURCE%"
    ```

    c. Enter the Description field:

    ```
    IBM TXSeries: Invoking CICS translator (cicstran):
    $(SOURCE)
    ```

    d. Enter the Outputs field:

    ```
    $(ProjectDir)\$(SOURCE)
    ```

    e. Repeat this procedure (steps a - d) for all of the CICS source files as well. Ensure that you change the **SOURCE** environment variable in the Command Line field appropriately for each CICS source file.

    f. Click **OK**.

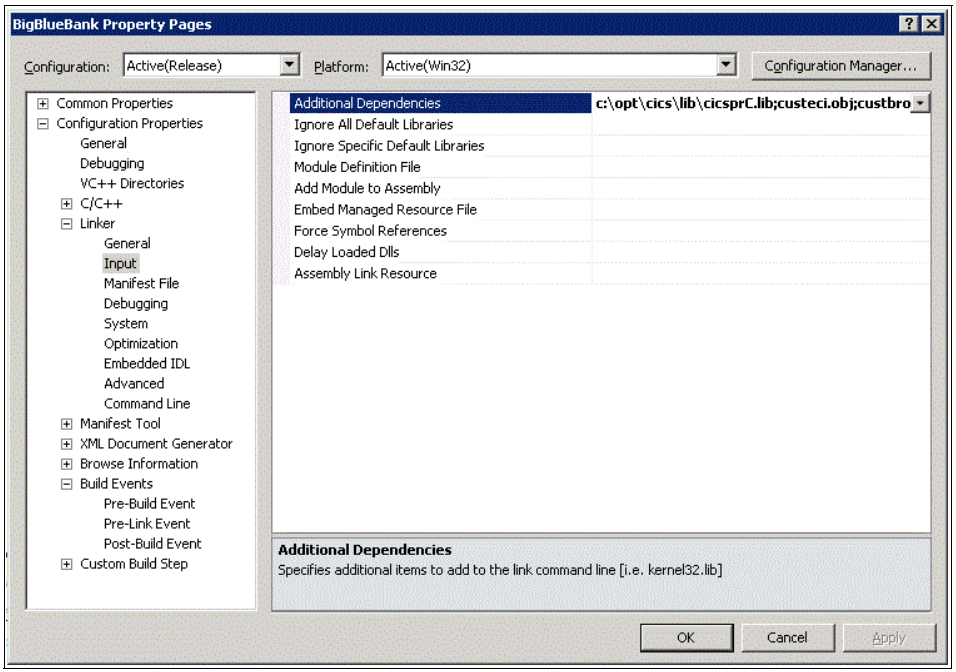


Figure 3-24 Custom build step window

11. Set the project properties to export the `main` function, and also to link with the object files to build the BigBlueBank module. Right-click the **BigBlueBank** project and select **Properties**.
12. This opens a new window with various configuration properties:
    a. Export the main function by selecting **Linker** and the suboption **Command Line**. In the Additional options field, enter the following data:

       ```
       /NODEFAULTLIB:LIBCMT.lib
       ```

    b. Add other input objects that have to be linked.
    c. Select **Linker** and the suboption **Input**. Enter the following information in the Additional Dependencies field:

       ```
       c:\opt \cics\lib\cicsprC.lib custeci.obj custbrowse.obj
       custget.obj custsignon.obj custupdate.obj
       ```

    d. Click **OK**.
13. You are now ready to build this solution. Select the menu option **Build** and the suboption **Build Solution** (F7 key). The build is now complete.

14. Figure 3-25 shows the Build output window.

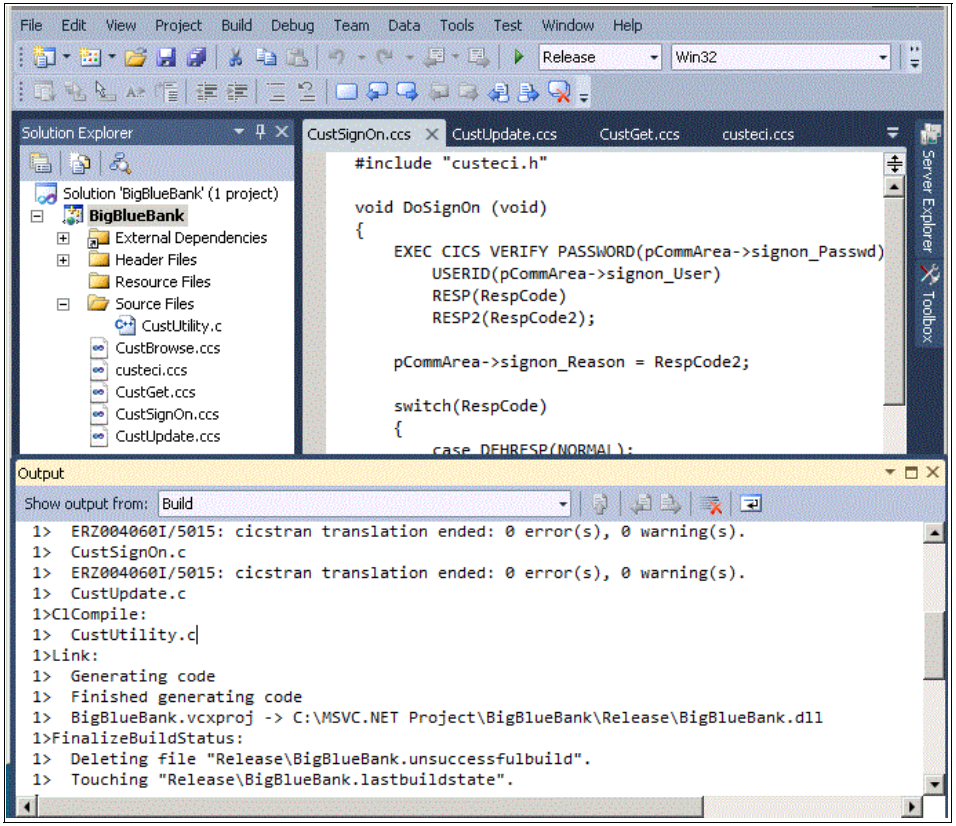


Figure 3-25 Build output window

Developing a sample BANKCRED application (Using CICS Foundation Class libraries)

To demonstrate CICS Foundation Class libraries, we implement the BigBlueBank system's Get Credit Rating (BANKCRED) application using CICS Foundation Classes. The BANKCRED application accepts a customer ID known to the bank, and returns the corresponding credit rating or score of that customer. The credit rating of each customer is on a mainframe system running CICS. We implement the following classes for this application:

bankcred.cpp This is the main application that receives the COMMAREA and returns the corresponding credit score for a given customer ID from a VSAM file.

bankcred.hpp This defines the BankCred main class and COMMAREA classes.

Compared to the normal C++ based CICS applications, C++ applications using CICS Foundation Classes do *not* contain the main function. The CICS Foundation Classes library provides a stub file called `iccmain.cpp`, which has the main function. The stub file is responsible for initializing the CICS Foundation Classes correctly, providing default exception handling, and releasing allocated memory after it is finished.

The sample application uses `IccBuf`, `IccKey`, `IccFile`, and `IccDataQueue` CICS Foundation Classes. The application operates on a COMMAREA, which is a sequence of characters (not numbers or binary data). The COMMAREA initially has a customer ID as the input, and the application browses through the VSAM file called BANKCRED, which has credit scores stored that correspond to each customer ID.

If the customer ID is present in the VSAM file, the corresponding credit score and a status value are returned back to the caller through the COMMAREA. If the customer ID is not present in the VSAM file, a status code of `1` (which indicates failure) is returned. The application uses the CSMT TDQ to log any success or failure messages.

Any TXSeries/C++ application using CICS Foundation Classes must include the `iccmain.cpp` file in its main source file. This file contains the run method implementation for the `IccUserControl` class, and `icceh.hpp` master header file, which includes all other CICS Foundation Classes header files. For a CICS Foundation Classes-based application, you do not have to run the CICS translator.

You either compile the application directly using the C++ compiler, or use the Microsoft Visual Studio or other such development tools to develop CICS Foundation Classes applications. In either case, you must specify the include path for the CICS Foundation Class header files during the compilation (`-I/usr/lpp/cics/include/icc`), and the CICS Foundation Class libraries (`-lcicsicc`) during linking.

3.8.6 Problem determination

We describe problem determination techniques in detail in Chapter 6, “Troubleshooting” on page 329.

3.8.7 Debugging

TXSeries provides a supplied transaction called CDCN, which is used to debug C++ applications. The CDCN transaction configures the IBM distributed debugger to debug C++ applications running under TXSeries. For more information about how to use the CDCN transaction for debugging C++ programs, see “Chapter 6, “Troubleshooting” on page 329.

3.9 Offline batch programming

We provide sample programs as part of the BigBlueBank application to provide sample reports. These are designed to show the techniques necessary to access SFS and relational database management system (RDBMS) data sources from outside of CICS.

3.9.1 Enabling IBM COBOL applications to access SFS files

To access SFS files from a COBOL application that runs on AIX, perform the following steps:

1. Compile and link the COBOL programs in the application by using the **`cob2_r4`** command.
2. Use the VSAM file system (**`FILESYS(VSA)`** is the default for the FILESYS runtime option).
3. Ensure that the SFS that your application will access is running.
4. Set the **`CICS_CDS_ROOT`** environment variable to the fully qualified name of the SFS. For example, the following code applies in this case:

   ```
   export CICS_CDS_ROOT=/.:/cics/sfs/test1
   ```

5. (Optional). If your application creates one or more SFS files, and you want to allocate the files on an SFS data volume that has a name other than `cicsSfsVol`, you can specify either or both of the following names:
   - The name of the SFS data volume on which the SFS files are to be created. To do so, assign a value to the **`CICS_SFS_DATA_VOLUME`** runtime environment variable. The data volume must be defined to the SFS. If you do not know which data volumes are available to the SFS, issue the **`sfsadmin list lvols`** command.

     By default, SFS files are created on the `cicsSfsVol` data volume.
   - The name of the SFS data volume on which alternative index files, if any, are to be created. To do so, assign a value to the **`CICS_SFS_INDEX_VOLUME`** runtime environment variable. The data volume must be defined to the SFS.

     By default, alternative index files are created on the same volume as the corresponding base files.
6. Identify each SFS file by using the fully qualified name:
   - The SFS name must precede the file name.
   - The alternative index file names must start with the base file name, followed by a semicolon (;) and the alternative index name.

     For example, if `/.:/cics/sfs/test1` is the SFS, and `VSAM04A` is an SFS file that has the alternative index `VSAM0A1`, you can identify `VSAM04A` by issuing the following export command:

     ```
     export
     VSAM04A="/.:/cics/sfs/test1/VSAM04A(/.:/cics/sfs/test1/VSAM04A;VS
     AM04A1)"
     ```

4

Enterprise integration

In this chapter, we look at IBM TXSeries from an integration perspective. We describe how TXSeries can access different enterprise systems, and how it can be accessed from both of the most-used client and server systems.

It is important to remember that TXSeries, like every IBM Customer Information Control System (IBM CICS) server, is able to access the CICS host naturally because of the intersystem communication (ISC) facilities that are a part of CICS itself.

This chapter provides information about the following topics:

4.1 TXSeries and relational database management systems

At this point of the book, you might be familiar with concepts, such as CICS resource definition and resource definition databases. Remember that each resource requires a resource definition to be used by CICS.

The link between CICS and a relational database management system (RDBMS) is obtained by using the extended architecture (XA) resource definition. Figure 4-1 shows a CICS server.

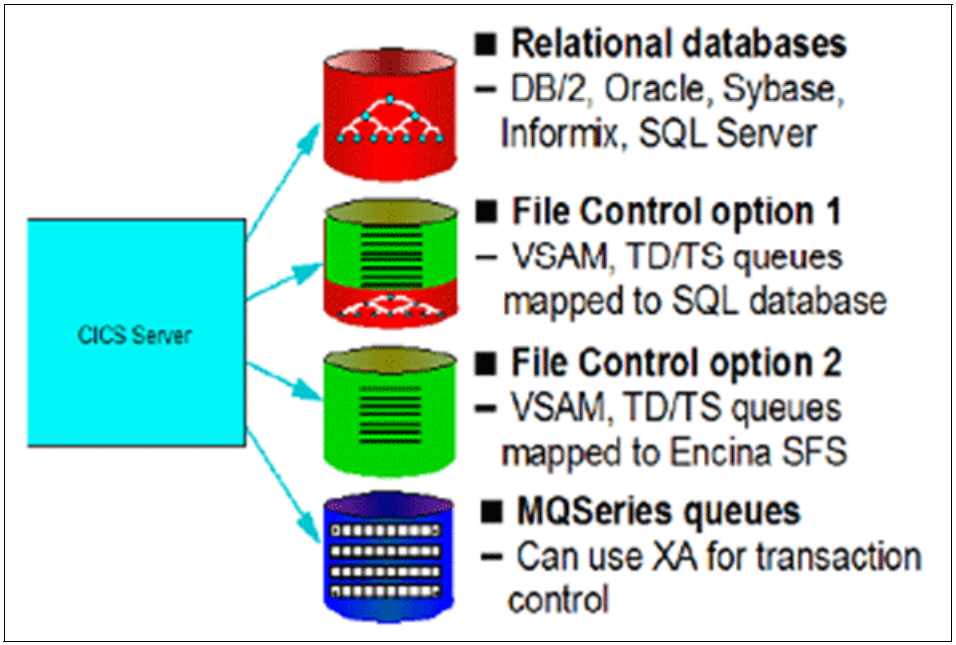


Figure 4-1 CICS server and XA resource managers

4.1.1 Configuring TXSeries for use with an RDBMS

CICS transactions can access relational databases by including embedded Structured Query Language (SQL) calls within the body of a CICS application program. Coordinated commitment and recovery of transactions that include both CICS and SQL calls are possible only with relational databases that support the X/Open XA interface.

Information: Releases of database products that are supported for versions of CICS are listed on the TXSeries support page on the following website:

http://www.ibm.com/software/htp/txseries/support

For the Common Business Oriented Language (COBOL) compiler that you are using, see the same web page to verify that your version is at the correct, supported level.

CICS fulfills the role of a transaction manager (TM) in the X/Open distributed transaction processing (DTP) model. See the X/Open XA specification publication for more details:

http://pubs.opengroup.org/onlinepubs/009680699/toc.pdf

CICS can coordinate distributed transactions to relational databases that support the XA interface. The XA interface is not an application programming interface (API). It is a system-level interface between a transaction manager and a resource manager (RM) in the X/Open DTP model. Nevertheless, application programmers have to be aware that they are coding in an XA-enabled environment.

For more information about application programming in such an environment, see the *Programming* task in the IBM TXSeries Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSAL2T/welcome

CICS provides sample programs of XA-enabled and non-XA-enabled environments in the `$CICS/examples/RM_support` directory. We strongly suggest that you use these sample programs to verify that your databases have been set up properly before attempting to use the databases in a production environment.

The instructions for XA-enabled and non-XA-enabled sample programs are in the `CHEESE.README` file. The `CHEESE.README` file describes an example database for a cheese company.

When the region is started in an XA transaction environment, the CICS application server makes an initial connection to the database using the XA open string. All CICS transactions run with either the default user cics or with a user who is explicitly specified in the XA open string.

At a minimum, the default user and the user identified in the XA open string must be granted database and table access to the RM. For a description of the switch-load file, see 3.5.3, “The switch-load file” on page 68.

Note: It is important to remember that CICS does not stop the region if the connection to an XA-enabled relational databases goes down. The region continues to be functional by default, and users must handle the SQL errors in their applications. `XAER_RMERR` is reported on the console when the database goes down.

4.1.2 Example of XA definition with TXSeries for IBM AIX

To define an XA-compliant RM to CICS, we use CICS commands. For more information about the full syntax of CICS commands, see the TXSeries documentation library:

http://www.ibm.com/support/knowledgecenter/SSAL2T/welcome

The following examples of commands show how to work with the XA resource manager product definitions (XAD):

- Add a new XAD. This example shows how to add an XAD entry for an XA definition named xaproduct:

  ```
  cicsadd -c pd -r regionA xaproduct attribute=attributeValue ...
  ```

- Change an existing XAD. This example shows how to change an XAD entry in the permanent database:

  ```
  cicsupdate -c xad -r regionA xaproduct attribute=attributevalue ...
  ```

- Delete an XAD. This example shows how to delete an XAD entry from the permanent database:

  ```
  cicsdelete -c xad -r regionA xaproduct
  ```

- View a product definition (PD) entry. This example shows how to view an XAD entry:

  ```
  cicsget -c xad -r regionA sales1 xaproduct
  ```

We have to know and supply the XA definition with both the RM-specific XA open string (the `XAOpen` attribute) and the location of the switch-load file (the `SwitchLoadFile` attribute):

- The `XAOpen` attribute of the XAD defines the text that is passed as an argument to the `xa_open` interface routine. You must provide the name of the database. Unless you also provide an operating system (OS) user ID and password, the region connects to the database under the default user name `cics`.

By default, the password is in plain text. To avoid storing the password in plain text, you must perform the following steps:

– Replace the password in the XAOpen with `@password@`.

  For example, enter the following **cicsadd** command:

```
cicsadd -cxad -r regionA XADEF
XAOpen="cicsdb2database,UID,@password@,TOC=p"
```

– Run **cicsxapasswd** to enter and store the password in a secured format:

```
cicsxapasswd -r regionA -p XADEF
```

  The cicsxapasswd is available from TXSeries V8.1 onwards.

- The `SwitchLoadFile` attribute of the XAD is used to specify the name of the switch-load file that lists the names of the XA routines. The syntax used by each RDBMS to identify the XA open string, and the procedure used to create the switch-load file, are specific for each RDBMS.

  For more information, see the Configuring task in the TXSeries documentation library. Additional XAD attributes are described under the *Administration reference* section of the *Reference* task in the TXSeries documentation library (click through to the appropriate product version):

  http://www.ibm.com/support/knowledgecenter/SSAL2T/welcome

The following list describes a typical `xa_open` string for various supported RDBMS resource managers:

- IBM DB2

```
cicsadd -c xad -r <region_name> DB2XAD SwitchLoadFile=db2xa
XAOpen="DB=dbname,UID=dbuser,PWD=dbpasswd,TOC=p"
```

- IBM Informix

```
cicsadd -c xad -r <region_name> INFXAD SwitchLoadFile=informxa
XAOpen="dbname"
```

- Oracle

```
cicsadd -c xad -r <region_name> ORAXAD SwitchLoadFile=oraclexa_dyn
XAOpen="Oracle_XA+ACC=P/dbuser/dbpasswd+SesTm=35"
```

- Sybase

```
cicsadd -c xad -r <region_name> SYBXAD SwitchLoadFile=sybasexa
XAOpen="-Udbuser -Pdbpasswd -Nconnection_name"
```

You can specify the dbpasswd as `@password@` and use the **cicsxapasswd** command to securely store the password.

You must set Thread of Control to `PROCESS (TOC=p)` when DB2 is configured as an XA resource or File Manager. The default `PROCESS` option forces the queue manager to serialize XA function calls so that only one call per process is made at any time.

Example for the AIX platform

Figure 4-2 shows the DB2 XA definition that we created for the demonstration environment of this book.

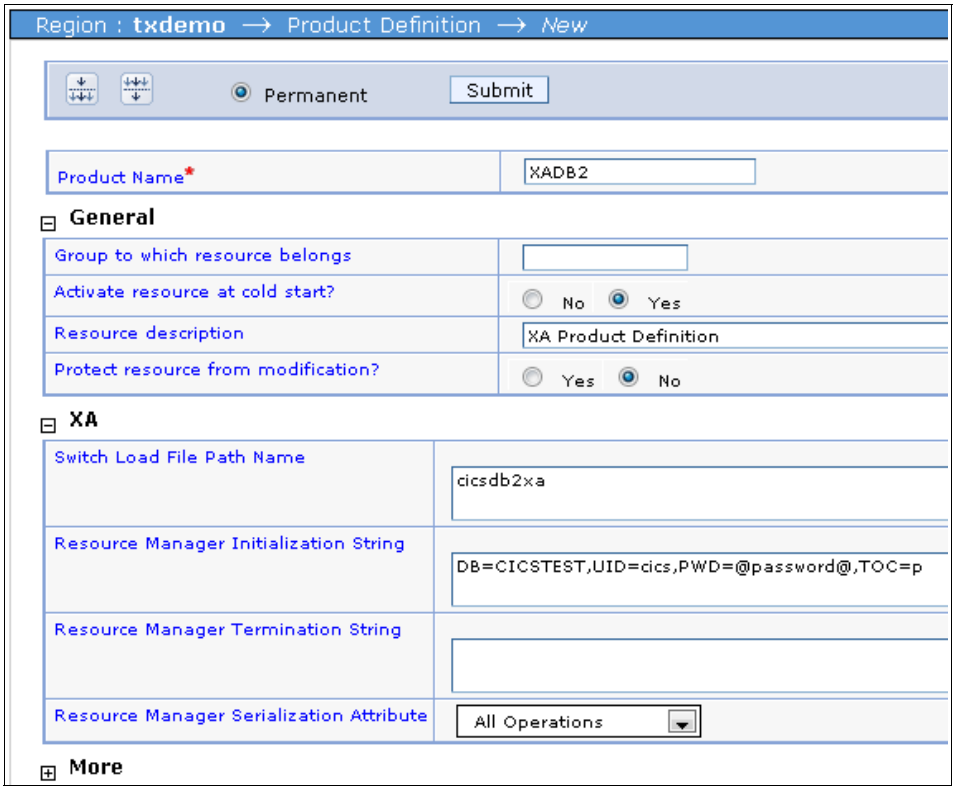


Figure 4-2 DB2 XA definition

Important: Remember to add the `DB2INSTANCE` variable (`DB2INSTANCE=<db2 instance name>`) to the environment file of your region (`/var/cics_regions/<RegionName>/<environment>`).

4.2 TXSeries and IBM WebSphere MQ

What we described in general terms for the XA definition in the previous RDBMS section is also valid for the XA connection to WebSphere MQ. Use WebSphere MQ V7.5 or later, to ask CICS to coordinate a global transaction. This includes the Message Queue Interface (MQI) calls that you have to set up an XA entry for in the queue manager in your CICS region.

4.2.1 Configuring TXSeries for use with WebSphere MQ

You can configure either an extended transactional client for use by CICS or a queue manager. In both cases, you have to add an XAD resource definition to your CICS region. You can do this by using the CICS resource definition online (RDO) **`cicsadd`** command. The XAD resource definition specifies the following information:

- An `xa_open` string
- The fully qualified path name of a switch-load file

The sources of the two types of switch-load files are provided by TXSeries in `amqczscix.c` (for a transactional client) and in `amqzscix.c` (for a queue manager) for AIX, Linux, Hewlett-Packard UNIX (HP-UX) on Intel Itanium, and Solaris, and in `amqzscin.c` for Microsoft Windows systems. Therefore, you can build your own switch-load file to use with TXSeries.

You can also build your own switch-load file if, for example, you do not want to use dynamic registration. Each switch-load file contains a function that returns a pointer to the XA switch structure that is used for dynamic registration, MQRMIXASwitchDynamic.

Figure 4-3 shows the XAD resource definition for WebSphere MQ for AIX that we created for this demonstration environment:

```
cicsadd -c xad -r txdemo MQXA \
ResourceDescription="MQM XA Definition" \
SwitchLoadFile="/var/cics_regions/txdemo/bin/amqzscix" \
XAOpen=mqdemo
```

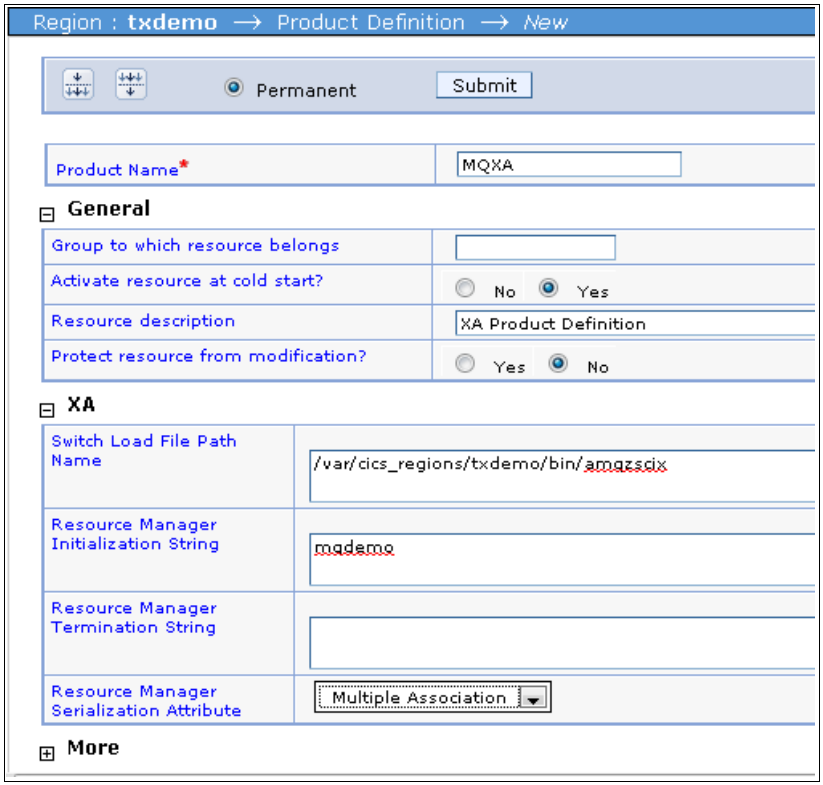


Figure 4-3 XAD resource definition

For more information about adding an XAD resource definition to a CICS region, see the *Configuring* topic in the previously linked TXSeries documentation library.

An important aspect about the TXSeries XA connection to WebSphere MQ is that you can add only one XAD resource definition for WebSphere MQ to a CICS region. Therefore, only one queue manager can be associated with a region, and all of the CICS applications that run in that region can connect only to that queue manager. If you want to run CICS applications that connect to a different queue manager, you must run the applications in a different region.

You can include calls to WebSphere MQ in a CICS transaction, and the WebSphere MQ resources are committed or rolled back as directed by CICS. Each application server in a region calls `xa_open` while it is initializing, and starts an MQI channel to the queue manager associated with the region.

You must start the queue manager *before* an application server starts, otherwise the `xa_open` call fails. All WebSphere MQ client applications processed later by the application server use the same MQI channel.

When an MQI channel starts and there is no security exit at the client end of the channel, the user ID that flows from the client system to the server connection message channel agent (MCA) is `cics`. Under certain circumstances, the queue manager uses this user ID for authority checks when the `MCAsubsequently` server connection attempts to access the queue manager's resources on behalf of a client application.

If this user ID is used for authority checks, you must ensure that it has the authority to access all of the resources that it needs to access. For detailed information about when the queue manager uses this user ID for authority checks, see the IBM MQ IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSFKSJ/mapfiles/product_welcome_wmq.htm

When you use the CICS two-phase commit process with WebSphere MQ, note the following requirements:

- WebSphere MQ and CICS must be on the same physical machine.
- WebSphere MQ supports CICS on a WebSphere MQ Extended transaction client.
- You must start the queue manager, with its name specified in the XAD resource definition field, before you attempt to start CICS.
- A CICS transaction must issue an **MQCONN** request before it can access WebSphere MQ resources.
- The **MQCONN** call must specify the name of the WebSphere MQ queue manager specified on the `XAOpen` entry of the XAD resource definition field for the CICS region. If this entry is blank, the **MQCONN** request must specify the default queue manager.
- A CICS transaction that accesses WebSphere MQ resources must issue an **MQDISC** call from the transaction before returning to CICS. Failure to do this might mean that the CICS application server is still connected, leaving queues open.

> **Requirement:** When the dynamic switchload file of WebSphere MQ is used, the region does not automatically connect back if WebSphere MQ stops and starts. The region has to be restarted for the existing WebSphere MQ connections from the region to work.

- You must ensure that the CICS user ID (`cics`) is a member of the `mqm` group, so that the CICS code has the authority to call WebSphere MQ.
- Create a system user called `CICSUSER` and add this user to the `mqm` group, so that the CICS transactions have the authority to use WebSphere MQ.
- For transactions running in a CICS environment, the queue manager adapts its methods of authorization and determining context:
  - The queue manager queries the user ID under which CICS runs the transaction. This is the user ID checked by the object authority manager, and is used for context information.
  - In the message context, the application type is `MQAT_CICS`.
  - The application name in the context is copied from the CICS transaction name.

See the IBM WebSphere MQ documentation on the following website for information about product installation, queue manager, and queues creation:

http://www.ibm.com/software/integration/wmq/library/

4.2.2 Deploying WebSphere MQ trigger monitor in CICS

The `accchgal.ccs` sample program is included with this book to demonstrate the WebSphere MQ trigger functionality. This section describes the WebSphere MQ trigger monitor supplied by CICS, also known as the CICS MQ Adapter task initiator (CKTI).

CKTI is the trigger monitor (or task initiator) supplied with TXSeries for Multiplatforms, and is used to start a transaction when the trigger conditions on any of its associated queues are met. CKTI starts a TXSeries transaction when a WebSphere MQ trigger message is put onto a specific queue, called the *Initiation Queue*.

Configuring CICS to use CKTI

To configure CICS, run the following **cicsadd** command:

```
cicsadd –c mqd –r txdemo -P "MQCN"
   ResourceDescription="MQCON CKTI Definition"
   MQMgrName="mqdemo"
   MQInitiationQueName="CICS.INITQ"
   MonitorTrigger=yes
   AutoStart=yes
```

This adds a stanza entry in the MQD stanza. After the cold start of the region, the CKTI transaction will be autostarted. This transaction monitors `CICS.INITQ` for any incoming messages. CKTI starts ACAL (as shown in Example 4-2) whenever it finds a message on `INITQ`.

Configuring on CICS

To configure the sample program on CICS, complete the following steps:

1. Compile the sample code with the command shown in Example 4-1.

   Example 4-1 Compiling the sample code

   ```
   #cicstran -lC accchgal.ccs
   #xlc_r -emain -bI:/usr/lpp/cics/lib/cicsprC.exp \
   -I/usr/lpp/cics/include \
   -I/usr/mqm/inc \
   -L/usr/mqm/lib -lmqm_r -L/usr/lpp/cics/lib \
   -oaccchgal accchgal.c
   ```

2. Copy the program to the region binary directory:

   ```
   #cp accchgal /var/cics_regions/txdemo/bin
   ```

3. Update the CICS region database, as shown in Example 4-2.

   Example 4-2 Updating CICS region database

   ```
   #cicsadd –c pd –r txdemo ACCCHGAL PathName=”accchgal” RSLKey=public
   #cicsadd –c td –r txdemo ACAL ProgName="ACCCHGAL" RSLKey=public
   RSLCheck=none
   ```

4. Add the following environment variable in the region environment file:

   ```
   MQ_INIT_QUEUE=CICS.INITQ
   MQ_QMGR=mqdemo
   ```

5. Start the region (from an unpowered state).

Configuring on WebSphere MQ

To configure the queue manager, complete the following steps:

1. Configure the `mqdemo` queue manager with the script commands shown in Example 4-3.

   Example 4-3 Script commands to configure the queue manager

   ```
   DEFINE QL(CICS.INITQ)
   DEFINE PROCESS(CICS.PROCESS) APPLICID(ACAL) APPLTYPE(CICS)
   ALTER QL(CICS.LOCAL.QUEUE) TRIGGER INITQ(CICS.INITQ)
   ```

2. To administer CKTI, TXSeries provides another transaction, CKQC:
   - To start CKTI manually, you can run the following command on cicslterm:

     ```
     CKQC STARTCKTI CICS.INITQ
     ```

   - To stop CKTI manually, run the following command on cicslterm:

     ```
     CKQC STOPCKTI CICS.INITQ
     ```

   - To display the status of CKTI, run the following command on cicslterm:

     ```
     CKQC DISPLAY
     ```

4.3 TXSeries and CICS host

As already mentioned in the introduction to this chapter, TXSeries can access and be accessed by the CICS host in an easy and natural way, where natural often means a transparent way. Figure 4-4 on page 129 shows TXSeries being accessed by the CICS host.

When TXSeries is part of a multiple system environment, the CICS regions can communicate with other systems. By doing so, CICS can provide users of the local region with services that are held on remote systems, or provide users on remote systems with services that are held on the local region.

This communication is achieved by networking the systems so that they can cooperate directly, enabling data and applications to be shared. The communication between the interconnected systems is referred to as *intersystem communication* (ISC).

Figure 4-4 shows the CICS host accessing TXSeries.

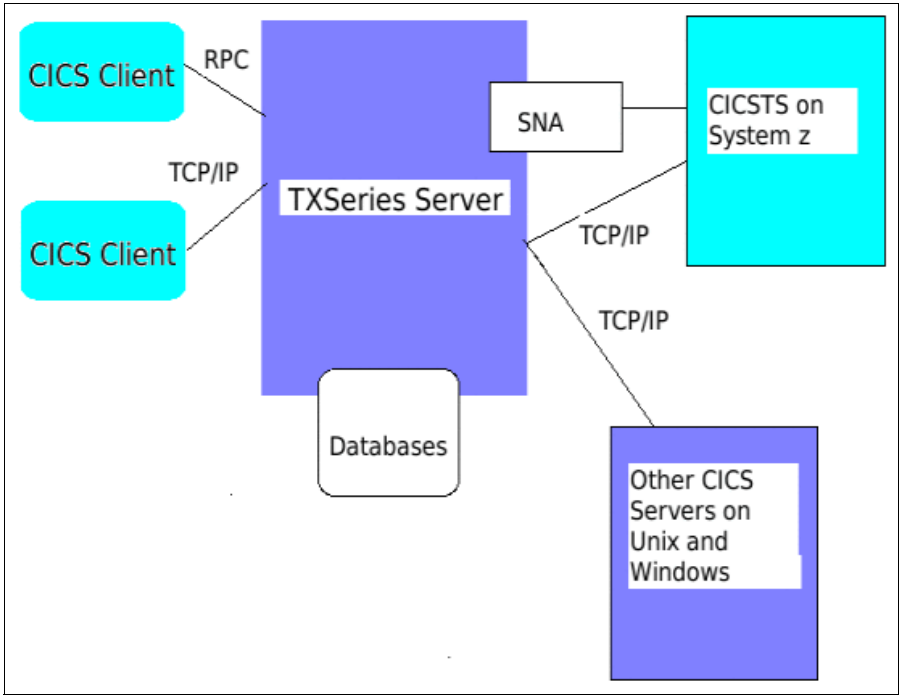


Figure 4-4 TXSeries accessed by CICS host

4.3.1 TXSeries intersystem communication facilities

The CICS ISC facilities, which TXSeries supports, simplify the operation of distributed systems. In general, this support extends the standard CICS facilities (such as reading and writing to files and queues), so that applications or users can use resources on remote systems without having to know where the resources are.

TXSeries supports the following CICS ISC facilities:

Distributed program link (DPL)

Extends the use of the EXEC CICS LINK command to enable a CICS application program to link to a program that resides on a different CICS system.

Function shipping

Enables an application program to access files, transient data queues, and temporary storage queues belonging to another CICS system.

Transaction routing Enables the execution of a transaction on a remote system. The transaction can display information about your terminal as though it is running on your local system.

Asynchronous transaction processing

Extends the `EXEC CICS START` command to enable an application to initiate a transaction to run on another CICS system. As with standard `EXEC CICS START` calls, the transaction requested in the `START` command runs independently of the application issuing the `START` command.

Distributed transaction processing (DTP)

Uses additional `EXEC CICS` commands that enable two applications running on different systems to pass information between them. These `EXEC CICS` commands map to the logical unit 6.2 (LU 6.2) mapped conversation verbs defined in the Systems Network Architecture (SNA).

Important: It is important to note that DTP is the only CICS intercommunication facility that can be used to communicate with applications not based on CICS. These applications must use Advanced Program-to-Program Communication (APPC) protocol.

As shown in Figure 4-4 on page 129, TXSeries-to-CICS host and CICS host-to-TXSeries communication can happen through SNA network protocol or Transmission Control Protocol/Internet Protocol (TCP/IP). You can also configure TXSeries to use IP interconnectivity (IPIC), which is based on TCP/IP, if the communication between the two happens through distributed program link (DPL) ISC facility.

TXSeries supports only DPL through IPIC. All ISC facilities are supported between TXSeries and CICS through SNA network. However, for TXSeries-to-TXSeries communications, the protocol used is TCP/IP. Although SNA configuration is also possible, usually this is not used due to its complexity and “heaviness” compared to TCP/IP.

Therefore, in the case of TXSeries-to-CICS host (or vice versa) communication, before starting with any CICS configuration either on TXSeries or on CICS host, you have to first configure SNA communication between the distributed and the mainframe system:

- Through local SNA support
- Through Peer-to-Peer Communications (PPC) Gateway server SNA support

See the specific documentation for your hardware platform for the SNA configuration part.

To use either of these implementation scenarios, you must configure various CICS resource definitions. You can do this by using the following tools:

- The IBM TXSeries web Administration console
- The command-line commands

Configuring resource definitions using one of these configuration methods is shown in the following implementation scenario section. Figure 4-5 shows the connection between distributed CICS and mainframe CICS through ISC.

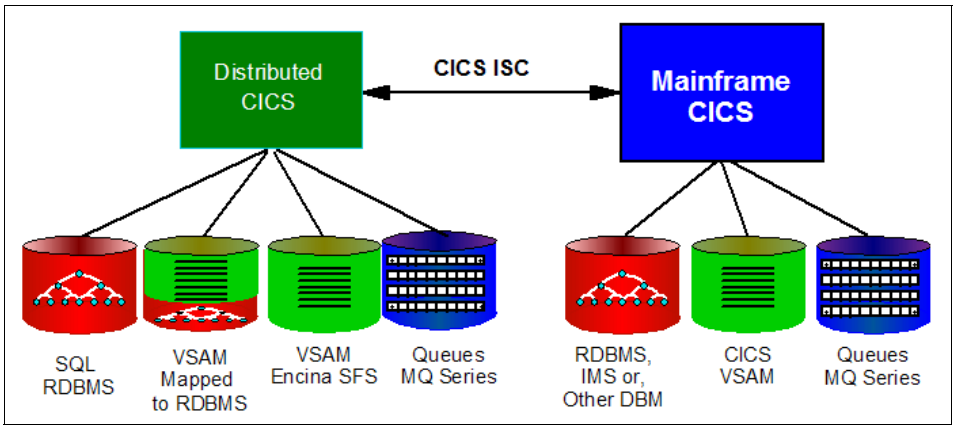


Figure 4-5 Distributed CICS connected to mainframe CICS using CICS ISC

4.3.2 TXSeries and local SNA configuration

As already mentioned, SNA protocol is supported by all of the members of the CICS Family. The local SNA support enables TXSeries CICS regions to communicate with every type of CICS product. In a local SNA configuration, synchronization levels 0 and 1 are supported. Therefore, in a distributed transaction TXSeries, CICS updates its own local resources and only accesses those belonging to the remote CICS in read mode.

If synchronization level 2 is required, you must use PPC Gateway server SNA support. In this case, TXSeries CICS has to manage a global transaction, and so update either only remote resources, or modify both remote and local ones.

The steps provided in the following section show how to create a local SNA CICS-to-CICS connection between a TXSeries for AIX and an IBM CICS Transaction Server (CICS TS) V3.1 host region.

On the AIX side, you must configure the following TXSeries resources:

- A listener definitions (LD) entry, which describes how a CICS region connects to a network. An SNA LD entry indicates that a CICS region uses local SNA. The LD causes a listener process to be started when the region starts.
- Two region definitions (RD) attributes:
  - LocalLUName: This identifies the LU name of the local CICS region
  - LocalNetworkName: This identifies the network to which it belongs. Optionally, we suggest that you specify a short name for the local region using the LocalSysId attribute.
- A communications definitions (CD) entry for each remote system with which the local region is to communicate.
- Optionally, you might have to configure the transaction definitions (TD) `TPNSNAProfile` attribute if both of the following conditions apply:
  - The local CICS region is a TXSeries CICS for AIX region.
  - Remote systems request transactions that are in the local region.

On the host side, you must configure the following CICS host resources:

- A CICS connection
- A CICS session

For additional information, see the CICS Transaction Server documentation on the following website:

http://www.ibm.com/software/htp/cics/library/

The following section provides the definitions that we created for the demonstration environment for both TXSeries for AIX and CICS TS V3.1. The same steps are also applicable for any later CICS TS version, such as versions 3.2, 4.1, 4.2, and 5.1.

For TXSeries

To configure CICS for local SNA by using TXSeries Administration Console, run the following procedure. We assume that default values are accepted for any attributes not discussed:

1. Log in to the TXSeries Administration graphical user interface (GUI) by entering the user ID and password. By default, the administration GUI opens on port 9080 on TXSeries V8.1. Check with your Administrator if the default configurations are modified:

   a. On TXSeries V8.1, use the following web address:

   `http://<server>:9080/txseries/admin`

   b. On TXSeries V7.1, use the following web address:

   `http://<server>/txseries/admin`

2. Configure an LD entry for txdemo region. On the left pane, select **Regions** → **txdemo** → **Listener** → **Add**. The Add Listener pane is displayed, as shown in Figure 4-6.

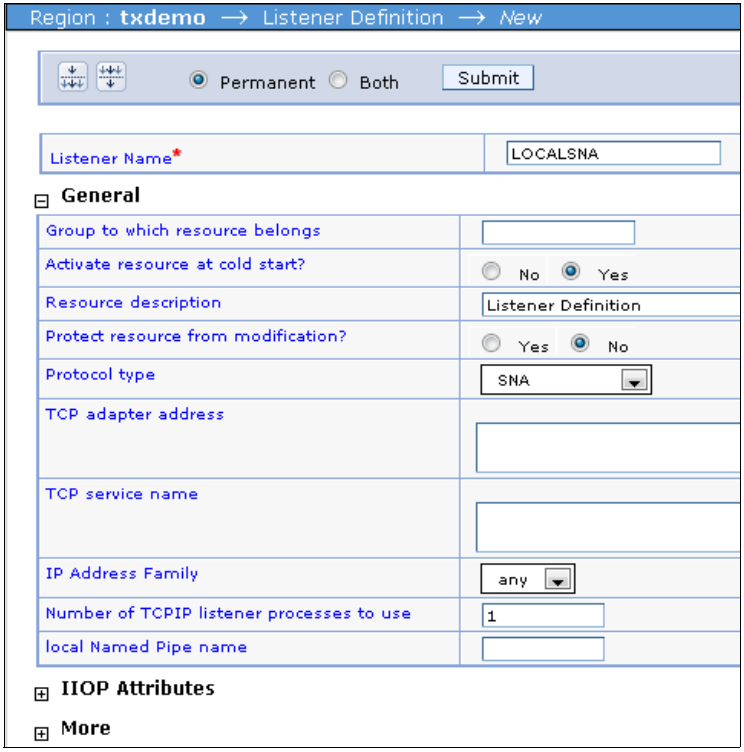


Figure 4-6 Adding the listener definition

3. Update the Region system identifier (short name) and Local LU name RD attributes, as shown in Figure 4-7.

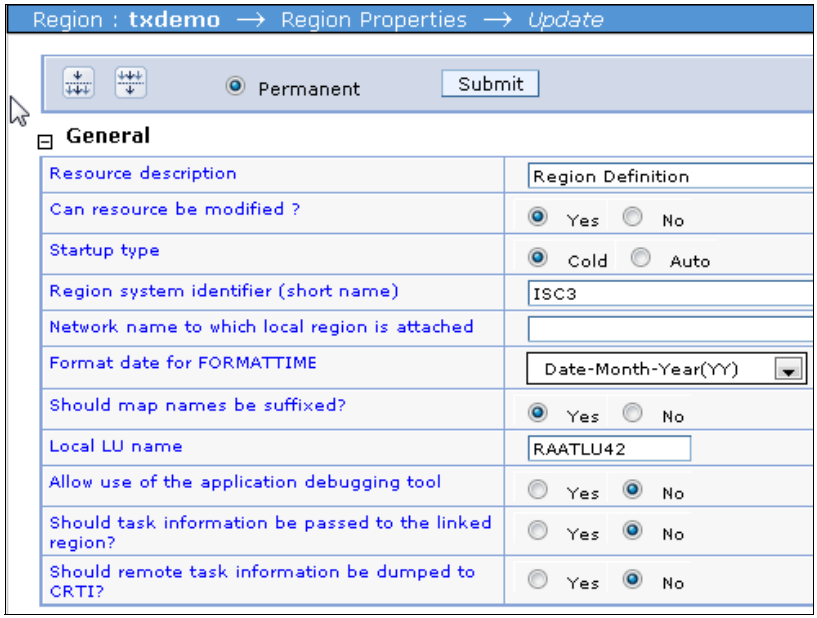


Figure 4-7 Updating the region identifier

4. Configure a CD entry for the CICS host, as shown in Figure 4-8 on page 135 and Figure 4-9 on page 136. On the left pane, choose **Regions** → **txdemo** → **Communication** → **Add**.

Figure 4-8 shows the General and SNA configuration for the CICS host.

Region : **txdemo** → Communication Definition → *New*

Permanent Both Submit

SYSID*	ISC2

General

Group to which resource belongs	
Activate the resource at cold start?	No Yes
Resource description	Communications Definition
Protect resource from modification?	Yes No
Connection type	CICS Local SNA
IP address family of the remote system	ipv4
Code page for transaction routing	IBM-037
Set connection in service?	Yes No
Is Task Information need to be passed to the linked region?	Yes No

SNA

Name of remote system	A6PO719L
SNA network name for the remote system	PSSCG3
SNA profile describing the remote system	
Default modename for a SNA connection	CICSISC0
Gateway Definition (GD) entry name	

Figure 4-8 Configuring the communication definition for the CICS host, Part 1

Figure 4-9 shows the TCP/IP and Security configuration for the CICS host.

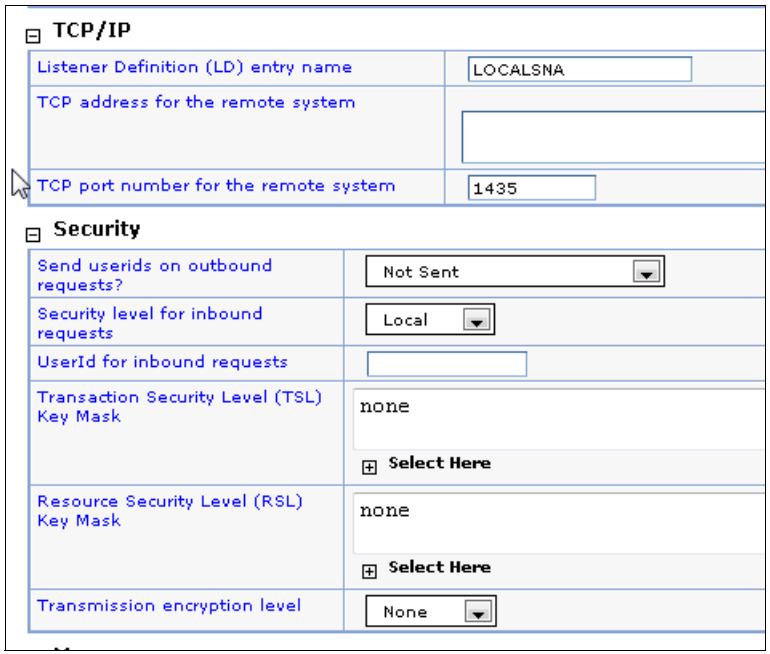


Figure 4-9 Configuring the communication definition for the CICS host, Part 2

For CICS Transaction Server

To configure a connection in CICS TS V3.1, follow these steps:

1. Configure a connection definition, as shown in Figure 4-10.

```
OBJECT CHARACTERISTICS                                        CICS RELEASE = 0640
 CEDA  View CONnection( PON1 )
  CONnection      : PON1
  Group           : ISC3CON
  DEscription     : CONNECTION FOR PSERIES CONNECTION
 CONNECTION IDENTIFIERS
  Netname         : RAATLU42
  INDsys          :
 REMOTE ATTRIBUTES
  REMOTESYSTem    :
  REMOTEName      :
  REMOTESYSNet    :
 CONNECTION PROPERTIES
  ACcessmethod    : Vtam              Vtam | IRc | INdirect | Xm
  PRotocol        : Appc              Appc | Lu61 | Exci
  Conntype        :                   Generic | Specific
  SInglesess      : No                No | Yes
  DAtastream      : User              User | 3270 | SCs | STrfield | Lms
  RECordformat    : U                 U | Vb

                                                  SYSID=9L11 APPLID=A6PO719L
```

Figure 4-10 Configuring the connection definition

2. Configure a session definition, as shown in Figure 4-11.

Region : **txdemo** → Communication Definition → *New*

Permanent Both Submit

SYSID*	ISC3

General

Group to which resource belongs	
Activate the resource at cold start?	No Yes
Resource description	Communications Definition
Protect resource from modification?	Yes No
Connection type	PPC Gateway
IP address family of the remote system	ipv4
Code page for transaction routing	%ASCII_CODEPAGE%
Set connection in service?	Yes No
Is Task Information need to be passed to the linked region?	Yes No

SNA

Name of remote system	APPO720L
SNA network name for the remote system	P55CG3
SNA profile describing the remote system	
Default modename for a SNA connection	CICSISC0
Gateway Definition (GD) entry name	GWY

Figure 4-11 Configuring a session definition

You have to ensure that the code page conversion exit within the CICS TS (DFHCNV) is correct for American Standard Code for Information Interchange (ASCII) and Extended Binary Coded Decimal Interchange Code (EBCDIC) conversion (8859-1/037).

3. In this case, add the following information (Figure 4-12), and assemble and link it into a library within the CICS DFHRPL.

```
DFHCNV TYPE=ENTRY,RTYPE=PC,RNAME=CREDITAP,USREXIT=NO,          X
      SRVERCP=037,CLINTCP=8859-1
DFHCNV TYPE=SELECT,OPTION=DEFAULT
DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=32767,    X
      LAST=YES
```

Figure 4-12 Adding DFHCNV

Information: In this demonstration environment, we decided to use the CICS host facility for the data conversion. It is also possible to ask TXSeries to convert the EBCDIC data received by the host. If you choose the second option, you have to define the DFHCNV macros and compile them using the TXSeries **`cicscvt`** command. For more information, see the topic about communicating in the TXSeries documentation library.

4.3.3 TXSeries and PPC Gateway server SNA configuration

A PPC Gateway server runs independently of a CICS region. A CICS region is connected to the PPC Gateway server using TCP/IP. The PPC Gateway server provides a link to the SNA network. To use PPC Gateway server SNA support, you must install and configure an appropriate SNA product on the same machine as the PPC Gateway server. The appropriate communications product depends upon the hardware platform in use.

A CICS region can use more than one PPC Gateway server. This type of configuration provides the following two performance benefits:

- If one PPC Gateway server fails, another is available as a backup.
- The processing load is spread across more than one PPC Gateway server and across multiple SNA products.

A single PPC Gateway server can also be shared by more than one CICS region. This can be an economical alternative if your CICS regions do not make many SNA intercommunication requests. However, such a configuration can overload a PPC Gateway server.

Every PPC Gateway server is associated with log volume. The log volume stores the server's log file, which contains its recoverable information. Usually, the PPC Gateway server runs a `cics` operating system (OS) user. As a result, the `cics` OS user must have read and write permission to this logical volume.

Information: Consider the following information before configuration:

- To use CICS commands to create or alter a PPC Gateway server, CICS must exist on the same machine as the server. To use CICS commands to communicate with a PPC Gateway server, CICS does not have to exist on the same machine. CICS does not support the use of a PPC Gateway server on Solaris. However, a CICS for Solaris region can use a PPC Gateway server that is running on a non-Solaris platform.
- CICS uses a gateway server definitions (GSD) entry to specify the characteristics required to start and stop the PPC Gateway server. This GSD entry is created automatically when the PPC Gateway server is created. It is deleted when the server is stopped. You can use several methods to install, configure, and run a PPC Gateway server. The following list shows the methods that are available.

The IBM TXSeries web administration console

This tool provides a GUI for configuring a PPC Gateway server. You can configure the PPC Gateway server using one of the following methods:

cicscp commands The cicscp configuration tool provides an easy-to-use command-line interface (CLI) that automates configuration as much as possible by using default values where necessary, and imposing some naming conventions.

CICS commands Low-level CICS commands enable the creation of more complex configurations than can be created using the cicscp configuration tool.

SMIT This tool provides a GUI method of configuring a PPC Gateway server on an AIX system.

ppcadmin commands These commands provide details about the status of transactions and the XLN process.

To create PPC Gateway definitions in this case, we use the TXSeries web administration console. The user must be logged in with cics administrator privileges to create the definitions using the following steps:

1. Decide on a one-character to eight-character `server_name` for the PPC Gateway server (in this example, `cicsgwy`).
2. To create a PPC gateway server, on the left pane choose **PPC Gateway Servers** → **All PPC Gateways** → **Create**. You see a window similar to Figure 4-13.

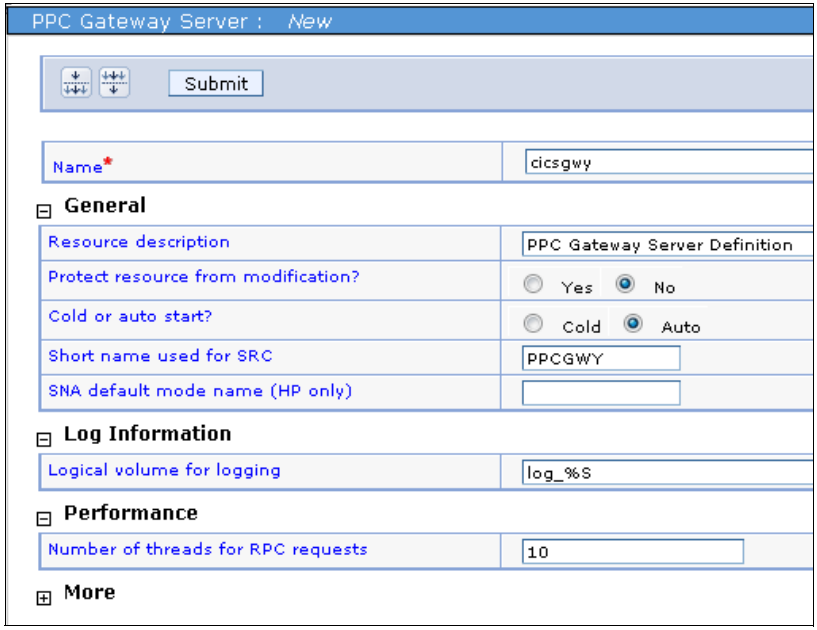


Figure 4-13 Creating a PPC Gateway server

3. Enter the PPC Gateway name and click **Submit**.
4. Associate the **PC gateway server** with a region (txdemo). Choose **Regions** → **txdemo** → **Gateway** → **Add and prepare the region definitions**.
5. Specify the following information, as shown in Figure 4-14 on page 141:
   a. Gateway Name (`GWY`)
   b. CICS PPC Gateway Server Name (`cicsgwy`)
   c. SNA LU name of the gateway (`RAATLU43`)
6. Click **Submit**.

7. Choose **Regions** → **txdemo** → **Region properties**, as shown in Figure 4-14.

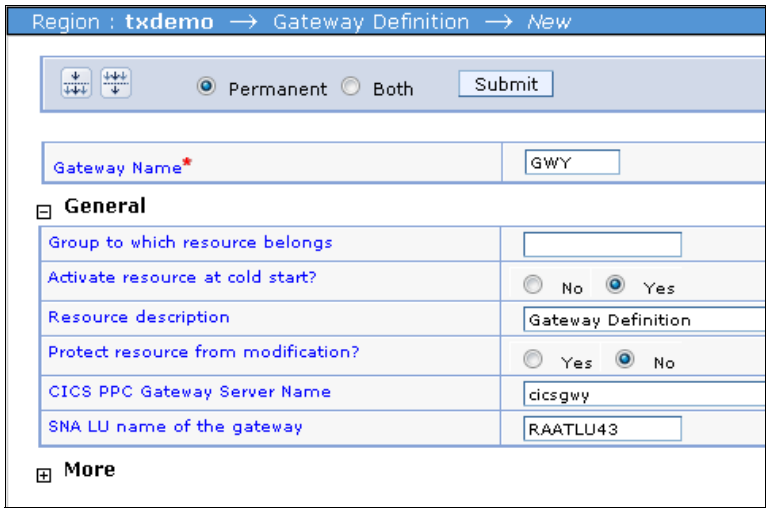


Figure 4-14 Creating Encina PPC Gateway server

8. Update the **Region system identifier (short name)** RD attribute, as shown in Figure 4-15.

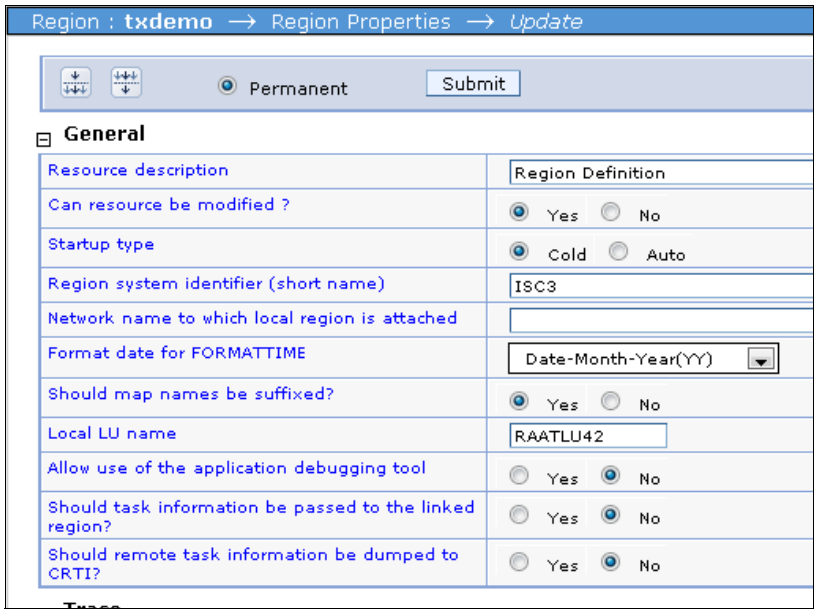


Figure 4-15 Adding the gateway

9. Set up the communication definition to communicate with the remote region. Use **Regions** → **txdemo** → **Communication definition** → **New** (Figure 4-16 on page 143):
   a. Enter the name of the communication identifier.
   b. Specify a **Connection type** of **PPC gateway**.
   c. Enter the LU Name of the remote system.
   d. Enter the name of the network to which the remote system is attached in the **SNA network name for the remote system** field.
   e. In the **Default modename for a SNA connection** field, enter the name of the SNA mode group to be used for intersystem requests in case an SNA mode name is not specified in either the PROFILE option of the **`EXEC CICS ALLOCATE`** command or in the `SNAModeName` attribute of the TD entry.

      If the default mode name for an SNA connection attribute is set to null (""), which is its default value, and the transaction has not provided a mode name, the mode name configured in the AIX SNA communications product's side information profile for the local LU name is used.
   f. Accept the default value or enter the appropriate value in the **Code page for transaction routing** field. This value determines what character set flows across the network during transaction routing. The correct code page depends on the *national language of your local region* and the *remote system type*.
   g. Accept the default value or select the appropriate value in the **Send user IDs on outbound requests** attribute to determine the type of security information (if any) sent with outbound requests.
   h. In the **Security level for inbound requests** field, select the type of security that you require:
      i. Accepting the default local option results in CICS running all incoming intersystem requests from the remote system under the user ID value that you specify in the **User ID for inbound requests** field. The user definitions (UD) entry for this user ID determines which resources these intersystem requests can access. This type of security is called *link security.*
      ii. Choosing the **verify** or **trusted** options results in CICS using the security information (such as the user ID and password) sent with the intersystem request. CICS also uses the user ID that you specify in the **UserID for inbound requests** field to restrict the resources that inbound intersystem requests can access. This type of security is called *user security.*

Region : **txdemo** → Communication Definition → *New*

Permanent Both Submit

SYSID*	ISC3

General

Group to which resource belongs	
Activate the resource at cold start?	No Yes
Resource description	Communications Definition
Protect resource from modification?	Yes No
Connection type	PPC Gateway
IP address family of the remote system	ipv4
Code page for transaction routing	%ASCII_CODEPAGE%
Set connection in service?	Yes No
Is Task Information need to be passed to the linked region?	Yes No

SNA

Name of remote system	APPO720L
SNA network name for the remote system	P55CG3
SNA profile describing the remote system	
Default modename for a SNA connection	CICSISC0
Gateway Definition (GD) entry name	GWY
Timeout on allocate (in seconds)	60

Figure 4-16 Adding the communication definition

Remember: Choose the **PPC Gateway** value for the Connection type field.

4.3.4 TXSeries communication to CICS host over IPIC

IPIC protocol enables TXSeries CICS regions to communicate with every type of CICS product. In the IPIC configuration, synchronization levels 0 and 1 are supported. Also note that, DPL is the only ISC facility supported by TXSeries over IPIC protocol.

Requirement: If the region needs to communicate with IBM CICS Transaction Server for z/OS over IPIC protocol, the region name should be in upper case.

The steps provided in the following section show how to create an IPIC connection between a TXSeries region and a CICS Transaction Server host region.

On the TXSeries region side, you must configure the following resources:

- A listener definitions (LD) entry for IPIC protocol. An IPIC LD entry indicates that a CICS region uses IPIC. This listener definition causes the `cicspc` listener process to be started at the region startup.
- Three region definitions (RD) attributes:
  - LocalLUName. This identifies the LU name of the local CICS region.
  - LocalNetworkName. This identifies the network to which the local CICS region belongs.
  - LocalSysId. A short name identifying the local CICS region.
- A communications definitions (CD) entry for each remote system with which the local region is to communicate.

For more information, see the topic about preparing a TXSeries region to communicate using IPIC with CICS TS under the *Communicating* task in the relevant TXSeries version IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSAL2T/welcome

On the host side, you must configure the following CICS host resources:

- A CICS IPCONN connection
- A CICS TCPIPSERVICE definition

See the topic about preparing CICS TS to communicate using IPIC under the *Communicating* task in the TXSeries IBM Knowledge Center.

The following section provides the definitions that we created for both TXSeries region and CICS TS V5.1.

For TXSeries

To configure CICS for IPIC by using the TXSeries Administration Console, run the following procedure. We assume that default values are accepted for any attributes not discussed:

1. Log in to TXSeries Administration Console.
2. Select the region (for example, `IPICREG1`) in the left pane, and then select **Listener**.
3. Provide details, such as Listener Name, TCP adapter address, and TCP service name.

4. Select a Protocol type of **IPIC Protocol**, as shown in Figure 4-17. Click **Submit** to save the listener definition.

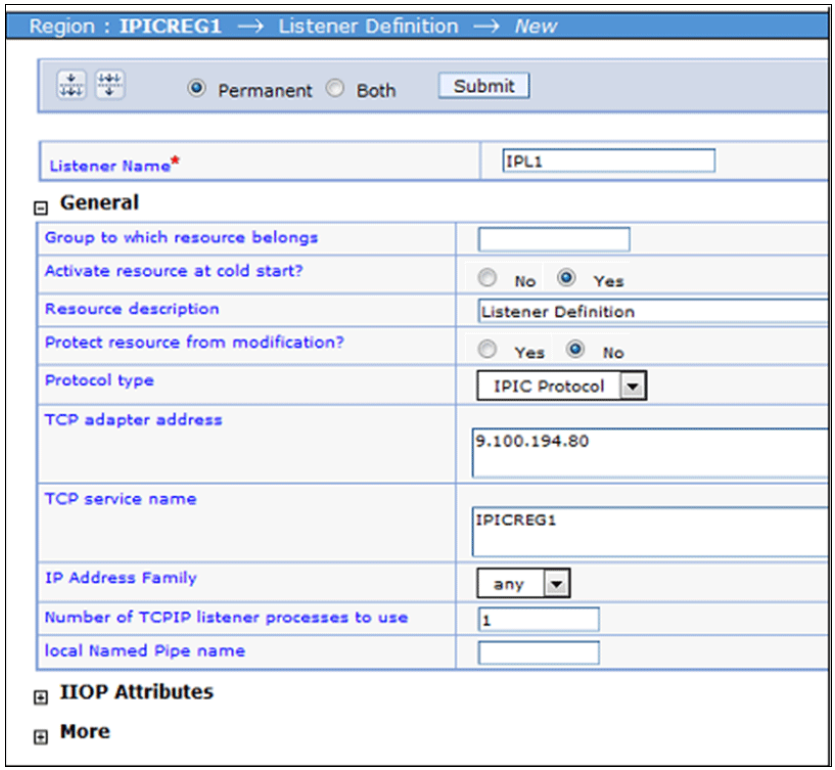


Figure 4-17 Adding the listener definition for IPIC

In Figure 4-17, IPICREG1 is the region name, IPL1 is the Listener Name, and IPICREG1 is the TCP service name.

5. Update the region definition for the LocalSysId, LocalNetworkName, and LocalLUName parameters, as shown in Figure 4-18. Click **Submit**.

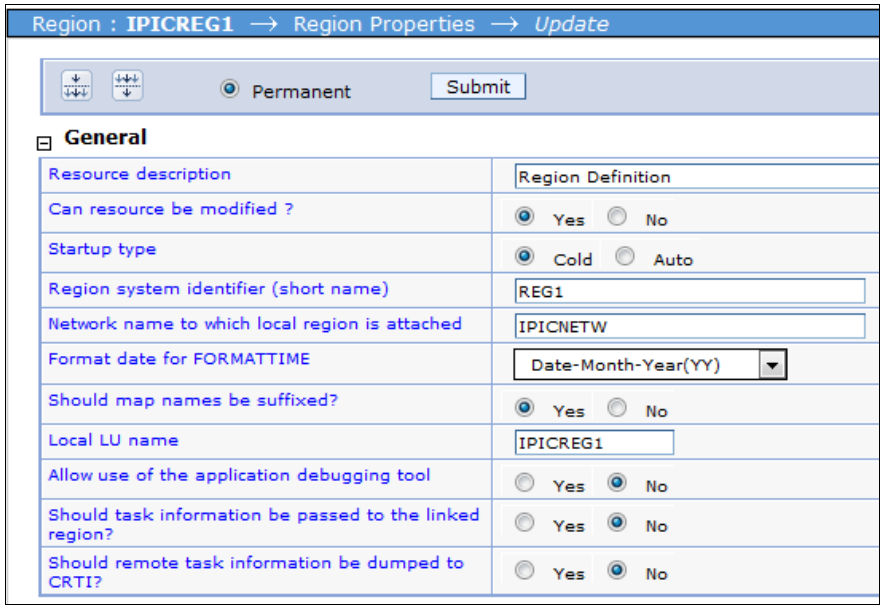


Figure 4-18 Updating Region Properties

In the example shown in Figure 4-18, REG1 is the value provided for LocalSysId, IPICNETW is the value provided for LocalNetworkName, and IPICREG1 is the value provided for LocalLUName.

6. Add a Communications Definition (CD), as shown in Figure 4-19. Click **Submit** to save the definition.

SYSID*	REG2
General	
Group to which resource belongs	
Activate the resource at cold start?	No Yes
Resource description	Communications Definition
Protect resource from modification?	Yes No
Connection type	CICS IPIC
IP address family of the remote system	ipv4
Code page for transaction routing	%ASCII_CODEPAGE%
Set connection in service?	Yes
Is Task Information need to be passed to the linked region?	Yes No
SNA	
Name of remote system	IPICREG2
SNA network name for the remote system	IPICNETW
SNA profile describing the remote system	
Default modename for a SNA connection	
Gateway Definition (GD) entry name	
Timeout on allocate (in seconds)	60
TCP/IP	
Listener Definition (LD) entry name	IPL1
TCP address for the remote system	9.100.194.81
TCP port number for the remote system	10202

Figure 4-19 Adding communication definition

In the example shown in Figure 4-19, REG2 is the name of the CD entry, Connection type is selected as CICS IPIC, IPICREG2 corresponds to RemoteLUName, IPICNETW corresponds to RemoteNetworkName, and IPL1 is the local CICS region's listener name. Also see that values need to be provided for RemoteTCPAddress and RemoteTCPPort attributes.

7. The TXSeries CICS region is ready with all of the configuration required for IPIC.

For CICS Transaction Server

To configure the CICS host region for IPIC communication, perform the following steps:

1. Connect to the CICS host region using appropriate tools. For details, see the topic about preparing CICS TS to communicate using IPIC under the *Communicating* task in the TXSeries V8.1 IBM Knowledge Center.
2. Create a TCPIPSERVICE definition using **CEDA** commands. For example, issue the following command:

   ```
   CEDA DEFINE TCPIPSERVICE(TCPIPSRV) GROUP(GROUP1)
   ```

   Figure 4-20 shows the CEDA window for the `TCPIPSERVICE` definition.

```
 OVERTYPE TO MODIFY                                          CICS RELEASE = 0680
  CEDA  DEFine TCpipservice( TCPIPSRV )
   TCpipservice    : TCPIPSRV
   GROup           : GROUP1
   DEScription   ==>
   Urm           ==> DFHISAIP
   POrtnumber    ==> 10202              1-65535
   STatus        ==> Open               Open | Closed
   PROtocol      ==> IPic               Http | Eci | User | IPic
   TRansaction   ==> CISS
   Backlog       ==> 00001              0-32767
   TSqprefix     ==>
   Host          ==> ANY
   (Mixed Case)  ==>
   Ipaddress     ==> ANY
   SOcketclose   ==> No                 No | 0-240000 (HHMMSS)
   MAXPersist    ==> No                 No | 0-65535
   MAXDatalen    ==>                    3-524288
+ SECURITY
  I New group GROUP1 created.
                                                   SYSID=REG2 APPLID=IPICREG2
  DEFINE SUCCESSFUL                          TIME: 23.08.17  DATE: 10/16/14
PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 4-20 Adding TCPIPSERVICE definition

3. Create an IPCONN definition using a **CEDA** command. For example, issue the following command:

   ```
   CEDA DEFINE IPCONN(REG1) GROUP(GROUP1)
   ```

Figure 4-21 shows the CEDA window for the `IPCONN` definition.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0680
 CEDA  DEFine Ipconn( REG1     )
  Ipconn          : REG1
  Group           : GROUP1
  DEScription  ==>
 IPIC CONNECTION IDENTIFIERS
  APplid       ==> IPICREG1
  Networkid    ==> IPICNETW
  Host         ==> 9.100.194.80
  (Mixed Case) ==>
  Port         ==> 10201              No : 1-65535
  Tcpipservice ==> TCPIPSRV
 IPIC CONNECTION PROPERTIES
  Receivecount ==> 010                1-999
  SENdcount    ==> 010                0-999
  Queuelimit   ==> No                 No : 0-9999
  MAxqtime     ==> No                 No : 0-9999
 OPERATIONAL PROPERTIES
+ AUtoconnect  ==> No                 No : Yes

                                                SYSID=REG2 APPLID=IPICREG2
 DEFINE SUCCESSFUL                        TIME: 23.35.24  DATE: 10/16/14
```

Figure 4-21 Adding IPCONN definition

4. Install the TCPIPSERVICE and IPCONN definitions using the following command:

   `CEDA INSTALL GROUP(GROUP1)`

Checking the status of the connection

To check the connection status, perform the following steps:

1. Start the IPICREG1 TXSeries region from an unpowered state.
2. Install the local region terminal and issue the following transaction:

   `CACQ SYSID=REG2`

   Here, REG2 is the CD entry name, which points to the connection to the CICS host region.
3. You get the following message at the terminal when the IPIC connection is acquired:

   `SYSTEM IS ALREADY IN ACQUIRED STATUS WITH SYSID 'REG2'`

For details, see the topic about the transaction to acquire an IPIC connection from the local TXSeries region to the remote region under the *Troubleshooting* task in TXSeries V8.1 IBM Knowledge Center.

You can also check the status of the IPIC connection from the CICS host region, by checking the status of the IPCONN.

After the connection is successful, you can attempt DPL facility across TXSeries and CICS host regions.

4.4 TXSeries and TXSeries

So far we have covered only the TXSeries-to-CICS host integration. In this section, we describe how TXSeries can communicate with another TXSeries region. Although it is always a CICS-to-CICS connection, we distinguish between the two configurations because usually TCP/IP is used rather than the SNA protocol. Therefore, the environment setup is different. Figure 4-22 shows a CICS-to-CICS communication.

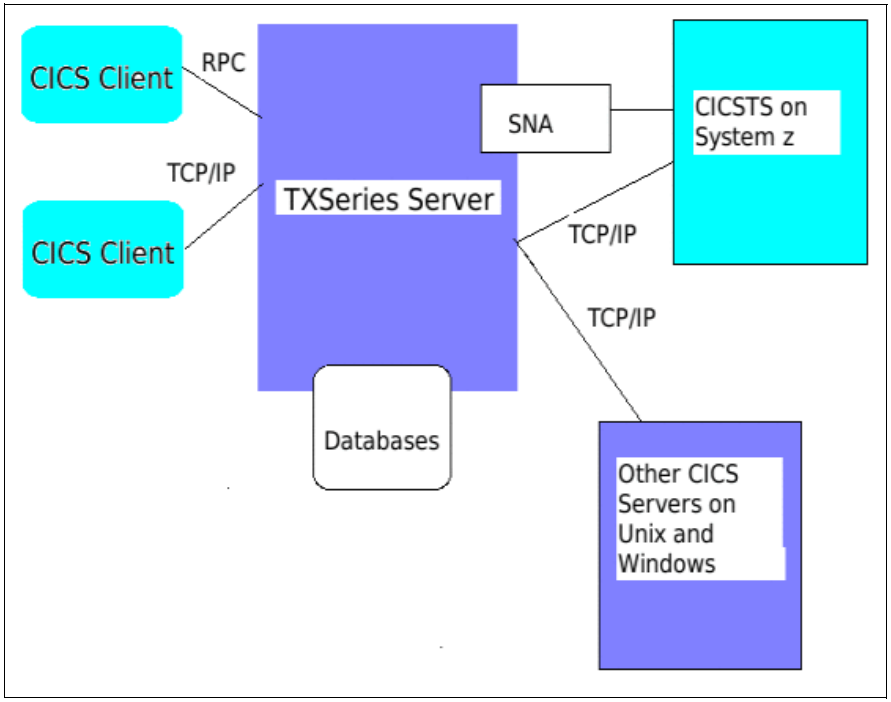


Figure 4-22 CICS to CICS connections

4.4.1 TXSeries and TCP/IP configuration

TXSeries CICS offers a choice of ways to use TCP/IP protocols:

- CICS Family TCP/IP support named `cics_tcp`, which enables connectivity to other TXSeries CICS, and IBM CICS Universal Clients.
- PPC TCP/IP support named `ppc_tcp`, which enables connectivity to other TXSeries CICS regions.
- IPIC support named `cics_ipic`, which enables connectivity to other TXSeries CICS regions, CICS Transaction Gateway (CICS TG) clients, and also with CICS Transaction Server for z/OS.

For introductory information about these ways of using TCP/IP, see the *Communicating* topic in the TXSeries documentation library. However, remember the following restrictions:

- All of the intersystem requests that flow on a CICS Family TCP/IP connection can use only synchronization level 0 or synchronization level 1.
- If you require synchronization level 2, you have to move to PPC TCP/IP support. However, many users decide to use PPC TCP/IP support as the default, and therefore also for synchronization levels 0 and 1.
- IPIC supports only synchronization levels 0 and 1 and the DPL facility.

4.4.2 TXSeries and CICS Family TCP/IP configuration

To configure CICS for TCP/IP, perform the following steps:

1. Identify a local name for your CICS region, which in an intercommunication environment is needed by remote systems to complete the following tasks:
   - Send your region intersystem requests.
   - Apply security checks and data conversion to intersystem requests received from your region.

   The name by which remote systems identify your region depends upon the communication method. The one- eight character name that you specify when you create your region is referred to as the *region name*, or the `APPLID`.
2. Set up an LD entry for your region.
3. Set up a CD entry for each remote system that your region is to communicate with, selecting **cics_tcp** as the Connection Type.

Information: Consider the following aspects of CICS configuration:

- On all platforms, CICS commands are used to configure the CICS LD and CD entries.
- Both the TCP adapter address and TCP service name attributes in the LD entry can be left blank:
  - A blank TCP adapter address indicates that CICS can use the address of any of the IP network adapters on the machine.
  - A blank TCP service name attribute means that CICS listens for connections on port 1435. CICS does not need an entry in `/etc/services` if TCP service name="" (blank). However, we suggest that you add an entry for the 1435 port to document that your region is using it. In addition, if you have more than one CICS region in your system, only one can use the default 1435 port, and for the others you have to define their TCP/IP ports explicitly.

4. Figure 4-23 shows the definitions created for the listener definition in this demonstration environment, including the following fields:

   a. Listener Name (`LDL1`)
   b. Protocol type (**TCP/IP**)
   c. TCP service name (`linksvc`)
   d. Number of TCPIP listener processes to use (1)

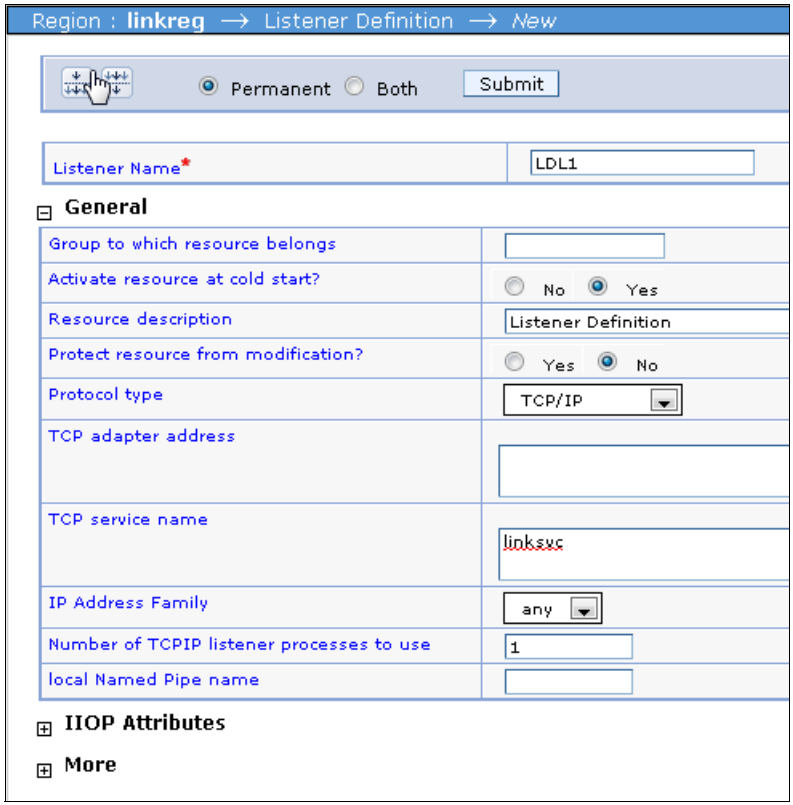


Figure 4-23 Adding the listener

5. Remember to define the `linksvc` service in `/etc/services` of your *IX system. In this case use the following values:

```
linksvc          20002/tcp        # listener port for ISC to pons2
```

> **Information:** On Windows, the services file is in the `C:\windows\System32\drivers\etc` directory.

6. Figure 4-24 shows the Add Communication pane, which includes the following values:

   a. SYSID (`TOC`)
   b. IP address family of the remote system (**ipv4**)
   c. Code page for transaction routing (`%ASCII_CODEPAGE%`)

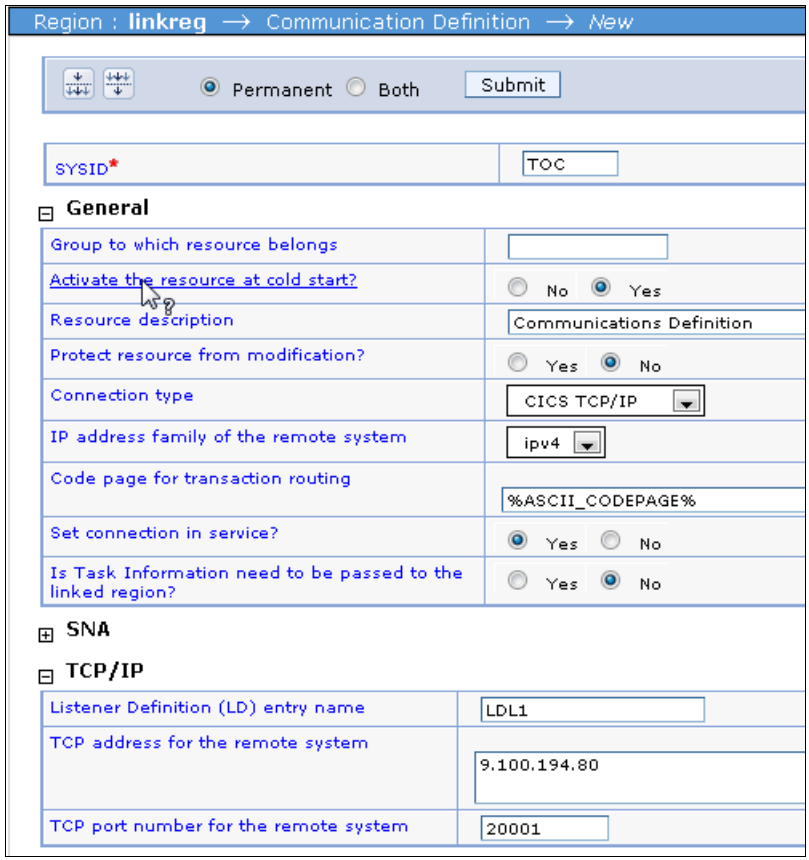


Figure 4-24 Adding the communication definition

In this example, the first TXSeries CICS region is called `linkreg`, and you have to contact a second TXSeries region named `hostreg`. (Note the value assigned to the **Name of remote system** field in Figure 4-19 on page 147, which in this case was `IPICREG2`).

7. Figure 4-25 shows the definitions necessary in the hostreg CICS region to talk to the linkreg CICS regions, including the following fields:

   a. Listener Name (`LDH1`)
   b. TCP service name (`hostsvc`)

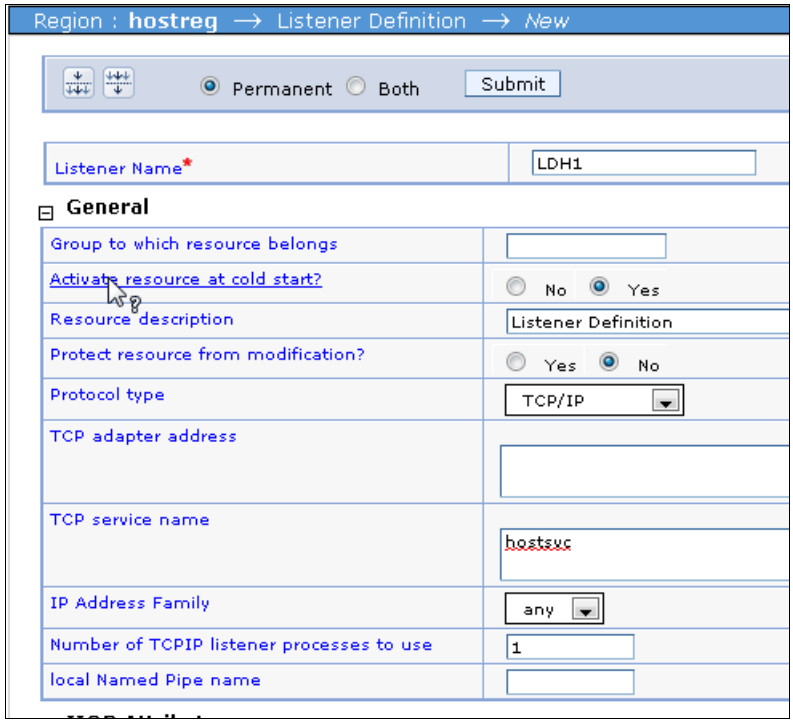


Figure 4-25 Adding the listener definition

8. You have to define the `hostsvc` TCP/IP service in the `/etc/services` file of your system:

```
hostsvc          20001/tcp
```

9. Figure 4-26 shows the Add communication pane:
   a. SYSID (`TOL`)
   b. Listener Definition (LD) entry name (`LDH1`)
   c. TCP address for the remote system (`9.100.194.80`)
   d. TCP port number for the remote system (`20002`)

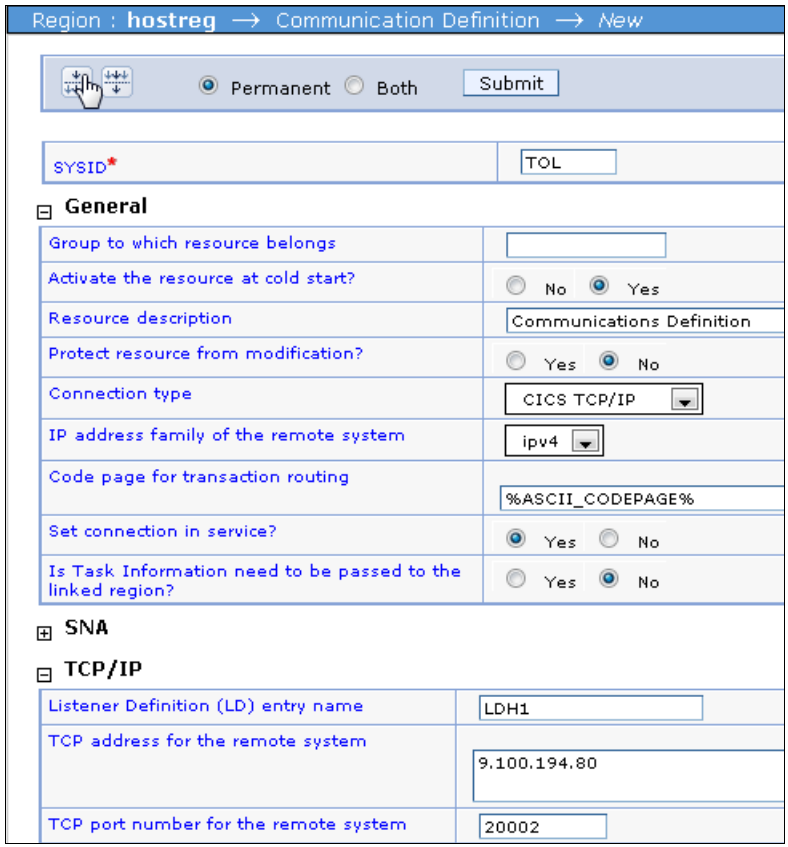


Figure 4-26 Adding the communication definition

Tip: If your local machine is an AIX system and it is managing many TCP/IP connections, it can run out of space in the `mbuf` pool. This is an area of memory allocated by TCP/IP when AIX is initialized. The default size is 2 megabytes (MB), which manages up to approximately 800 CICS Family TCP/IP connections.

You can display the size of the `mbuf` pool, along with other network options, by using the **`no -a`** command. The `mbuf` pool size is called *the wall*. It is expressed in multiples of 1024 bytes.

10. If you are using an AIX system, we check the `mbuf` pool size of the machine, as shown in Example 4-4.

Example 4-4 Checking mbuf pool size

```
{pons1:root}/ -> no -a | grep thewall
                  thewall = 1048576
```

11. You do not have to modify the `mbuf` pool size. However, the **`no -o thewall`** command enables you to alter its value, if necessary. Example 4-5 sets the `mbuf` pool size to 4096 x 1024 bytes (4 MB). You have to be the root user to issue this command.

Example 4-5 Command to change mbuf size

```
{pons1:root}/ -> no -o thewall 4096
```

The **`no`** command only operates on the current running kernel, and therefore you must rerun it each time your machine is started. To configure your machine to run this command automatically on startup, add the command to the `/etc/rc.net` file.

> **Attention:** Be careful when you use the **`no`** command, because it performs no range checking on the parameters you specify. If used incorrectly, the **`no`** command can cause the system to become inoperable.

TXSeries and PPC TCP/IP configuration

To configure a connection for PPC TCP/IP support you must complete the following steps:

1. Configure a CD entry for each remote region with the attribute `ConnectionType=ppc_tcp`.
2. Set the **`CICS_HOST`** environment variable if you are not using the Cell.

You must configure the **`CICS_HOSTS`** environment variable in your region environment file. The following examples are for UNIX and Windows:

- For UNIX environments and for a CICS region with the name `txdemo`, the file is called `/var/cics_regions/txdemo/environment`.
- For Windows environments and a CICS region with the name `cicstest`, the file is called `\var\cics_regions\cicstest\environment`.

The content of the **`CICS_HOSTS`** environment variable lists the hosts where the servers are located, as shown in the following example:

```
CICS_HOSTS='HOST1 HOST2 HOST3'
```

The region searches `HOST1` for string binding of the remote region. If it does not exist on that host, `HOST2` is searched, then `HOST3`. If you have not set **`CICS_HOSTS`**, the region assumes that the remote region (and the string binding) is on the local machine.

> **Tip:** In this case, an LD is not required because Remote Procedure Calls (RPCs) are used. These RPCs flow through the RPC Listener process, which is always present in a CICS region.

Using the same definition names created for the previous configuration, in case of `ppc_tcp` connection, the CD entry looks similar to Figure 4-27.

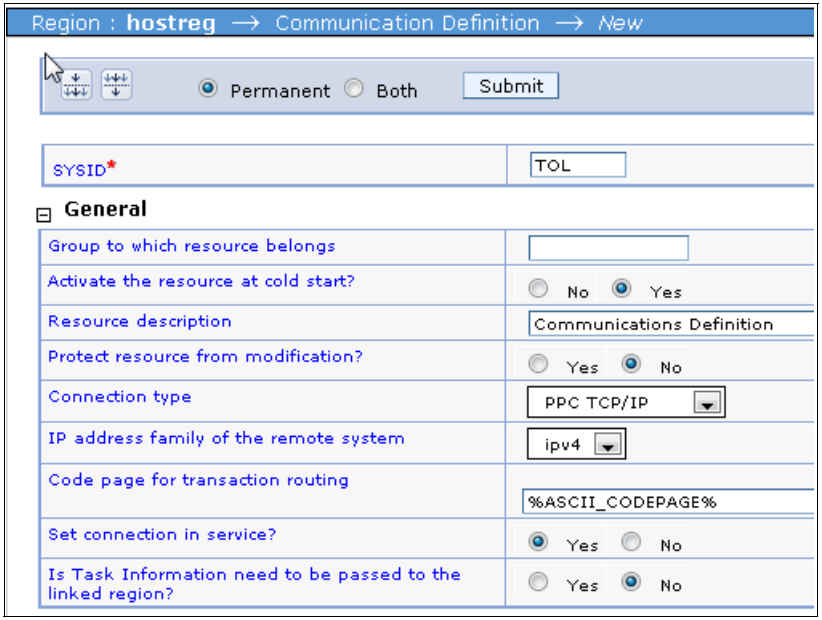


Figure 4-27 Adding the communication definition

4.5 TXSeries and intersystem security

The security requirements for a region that communicates with other systems are an extension of the security requirements for a single, stand-alone region. CICS security uses the concepts of user sign-on to provide a user the authority to access sensitive transactions and resources. These facilities are extended for intercommunication functions to include the remote users and remote systems.

CICS assumes that it is responsibility of every system to verify the authenticity of all of the requests that it receives. These inbound requests can be received from individual users or remote systems. The remote system can be another CICS system or a system not based on CICS.

Some initial effort is required to determine how to set up intercommunication according to the policies of your local system and the privileges that the users of your region require. However, after the security checking is set up, CICS security checking operates automatically, without the need for remote users or application programs to take specific security actions.

The security checks supported by TXSeries are based on SNA LU 6.2 security services. Therefore, they are available for controlling access from SNA-connected systems. Where appropriate, these services have also been extended to systems connected by TCP/IP.

When you configure your CICS system, in CD entries, you define all of the remote systems from which you want to accept intercommunication requests. You also have to define the security levels that you are prepared to give to those remote systems with the RemoteSysSecurity, LinkUserid, RSLKeyMask, and TSLKeyMask attributes in the CD entry.

However, before accepting requests, your system must first verify that the remote system is actually the system that it claims to be. This verification process is known as *authentication*. The mechanisms available to do this depend upon the type of your network connection.

The first security problem your CICS system has to resolve is to correctly determine the identity of the remote system that is attempting to initiate an intercommunication request. You must ensure that an alien remote system cannot impersonate another.

4.5.1 Authenticating systems across CICS TCP/IP connections

When an intersystem request is received on a CICS TCP/IP connection, your region cannot verify the identity of the remote system. There is no mechanism available that enables you to detect when an unauthorized system has deliberately impersonated another.

CICS can extract the IP address and port number of the remote system, but this is easy for an alien system to imitate. If you have defined an entry to enable CICS TCP/IP connections, unless your IP network is private and secure, define the security attributes in the CD entry on the assumption that the identity of the remote system has not been verified.

4.5.2 Authenticating systems across PPC TCP/IP connections

The identity of a system connected using PPC TCP/IP is determined by the `sarpcd` process provided RPC services using the **`CICS_HOSTS`** variable. Remote systems use an RPC to schedule an intersystem request with your CICS region.

4.5.3 Authenticating systems across SNA connections

When a remote system uses an SNA connection to communicate with your CICS system, it must first establish a session with it. This session is created by an exchange of flows called a *BIND*. You can associate a password with the BIND. This process is known as *bind-time security*, or *LU-LU verification*. It enables each system to verify the identity of the other.

These passwords are not sent between the two systems. Each system demonstrates its knowledge of the password by being able to correctly encrypt random numbers supplied by the partner, using the password as a key.

The bind is successful only when both systems can establish that they have the same password. Bind passwords are set up in the SNA product that is managing your SNA connectivity. See your SNA product documentation for a description of how to set the bind password for a connection.

4.5.4 Authenticating systems across PPC Gateway connections

If you are using the PPC Gateway server to support SNA connections to remote systems, CICS issues a DCE RPC to contact the PPC Gateway server. Similarly, the PPC Gateway server uses an RPC to schedule an inbound intersystem request from an SNA remote LU. Therefore, the region can verify that the intersystem request comes from a trusted PPC Gateway server.

4.5.5 Authenticating systems across IPIC connections

IPIC protocol can be used for communication between IBM CICS Transaction Gateway (CICS TG) client and TXSeries, between two TXSeries regions, and between a TXSeries region and a region on CICS Transaction Server. This communication can be made secure through different ways:

- Securing the communication channel using Secure Sockets Layer (SSL)
- Securing intersystem communication using the security configuration attributes in the communication definitions (CD) of TXSeries region

For details about how to configure security through these methods, see the topic about IPIC security under the Product overview task in the TXSeries V8.1 IBM Knowledge Center.

4.5.6 CICS link security

You can define security levels for the transactions and resources in your CICS system that apply to all intercommunication requests that are received from a particular system. This form of security is known as *link security*.

To use link security, you must have a security manager. TXSeries has its own security manager. However, if your operating system supports an external security manager that TXSeries supports, you can use that rather than, or in conjunction with, the CICS internal security manager.

An external security manager is a user-supplied program. It enables you to define your system's own security mechanism to prevent an unauthorized user from accessing resources from application programs and an unauthorized initiation of CICS transactions.The following description helps you to implement CICS internal security, which uses Transaction Security Level (TSL) and Resource Security Level (RSL) keys to restrict access.

Remember that the security keys defined for link security apply to all requests received from a particular remote system. Therefore, the list of security keys must include all of the keys required by every user from the remote system. If the needs of the users from a remote system vary, this list of security keys might provide more access to some users than is necessary.

If this is not acceptable, consider using the CICS user security instead. CICS user security enables you to set up security keys based not only on the system that sent the request, but also on the user associated with that request. We describe CICS user security in the following section.

If link security is sufficient, you can set it up using the following steps:

1. Set the CD `RemoteSysSecurity` attribute to `local` for the connection to the system for which you want link security implemented.
2. Specify a link user ID for the connection with the CD `LinkUserId` attribute. This user ID is associated with any request that is received from the remote system at the other end of the connection.
3. Create a UD entry for the connection's link user ID. Use the UD `TSLKeyList` and the `RSLKeyList` attributes to specify the transactions and resources that can be accessed by inbound requests.

An alternative implementation of link security is shown in the following steps:

1. Set the CD `RemoteSysSecurity` attribute to `local` for the connection to the system for which you want link security implemented.
2. Do `not` specify a link user ID for the connection with the CD `LinkUserId` attribute.
3. Use the CD `TSLKeyMask` and `RSLKeyMask` attributes to define the TSL and RSL keys to use for the connection.

Important: These two methods differ in the following ways:

- When a link user ID is specified for the connection, the user is logged on as the connection's link user ID, and the keys defined for the link user ID are used. *This is the preferred method.*
- When a link user ID is not specified for the connection, the user is logged on as the region's default user ID, and the connection's `TSLKeyMask` and `RSLKeyMask` keys are used. Any keys defined for the region's default user ID are ignored.

4.5.7 CICS user security

CICS user security provides a more granular security checking than link security. The security is more granular because it enables you to base the TSL and RSL keys that apply to inbound requests not only on the remote system but also on the user who originated the request.

This is achieved by setting up the remote system to flow the user ID of the user with the intersystem request. When the request is received from the remote system, your region provides access to any transaction or resource that matches the security keys that are defined for both the flowed user ID and the connections link user ID. These keys are defined with the UD `TSLKeyList` and `RSLKeyList` attributes.

> **Remember:** A link user ID is specified with the CD `LinkUserId` attribute. If a link user ID is not specified (that is, if the setting is `LinkUserId=""`), the keys defined with the CD `TSLKeyMask` and `RSLKeyMask` attributes are used in conjunction with the flowed user ID instead.

Combining the keys for the link with the keys for the user ensures that remote users with the same user ID as local users (or remote users from other systems) can be set up with different access privileges. Similar to the CICS link security, the CICS user security can also be implemented in two ways:

1. Ask the administrator of the remote system to generate the following information:
   a. A list of the user IDs of the remote users from the remote system who access your region, and the security keys that each of them require. Then, generate a list of security keys comprising all of the security keys required by these users. These keys are known as the *link keys* for the system.
   b. A list of those user IDs that appear in both the remote system and your local system, irrespective of whether they are used to access your region from the remote system. For each of these user IDs, check the level of security if they access your region from the remote system.

      To do this, compare the keys that are common to the local UD entry for the user ID with the link keys, as calculated in 1a. Then, look at the transactions and resources that these specifications give them access to.

      > **Tip:** If the results are unacceptable, either the keys assigned to your local transactions and resources must be changed or the user ID in one of the systems must be renamed.

2. Set up the remote system to flow user IDs.
3. Set the CD `RemoteSysSecurity` attribute to `trusted` or `verify` for the connection to the remote system.
4. Define UD for each of the user IDs identified in 1a.
5. Define a UD entry for the name of the remote system. This is important because some CICS tasks (which are initiated by the remote system) can run with the remote system name as the user ID. We advise that this user ID be used *only* for this purpose (that is, it is not used as a normal user ID). The default attribute values suffice for this UD entry.
6. Define a UD for the link user ID. This user ID represents the remote system.
7. Set the CD `LinkUserId` attribute to the user ID defined in step 5.

The following method is an alternative:

1. Ask the administrator of the remote system to generate the lists of user IDs, as described in 1a on page 162 and 1b on page 162 in the previous list.
2. Set up the remote system to flow user IDs.
3. Set the CD `RemoteSysSecurity` attribute to `trusted` or `verify` for the connection to the system for which you want user security implemented.
4. Define a UD for each of the user IDs identified in 1a on page 162 of the previous list.
5. Define a UD entry for the name of the remote system. This is important because some CICS tasks (which are initiated by the remote system) can run with the remote system name as the user ID. We advise that this user ID be used *only* for this purpose (that is, it is not used as a normal user ID). The default attribute values suffice for this UD entry.
6. Do *not* specify a link user ID for the connection with the CD `LinkUserId` attribute.
7. Use the CD `TSLKeyMask` and `RSLKeyMask` attributes to define the TSL and RSL keys to use for the connection.

In either case, you must have TSL and RSL keys assigned to the resources and transactions for which you want user security applied. For additional details, see documentation specific to CICS. In particular, see the topic about security in the TXSeries documentation library.

Information: The following differences exist between these two methods:

- When a link user ID is specified for the connection, the user is logged on as the flowed user ID, and the keys that are defined for both the link user ID and the flowed user ID are used. *This is the preferred method.*
- When a link user ID is not specified for the connection, the user is logged on as the flowed user ID, and the `TSLKeyMask` and `RSLKeyMask` keys that are defined for both the connection and the flowed user ID are used.

Setting the RemoteSysSecurity attribute to trusted or verify

The `RemoteSysSecurity` attribute can be set to either `trusted` or `verify`. Set the value to `trusted` when the following conditions are true:

- The remote system *does not* send passwords with user IDs. This applies to CICS host systems. TXSeries CICS systems might send passwords, but their default behavior is not to send them.
- The remote SNA system *does* send passwords, but the SNA product that you are using verifies the password and does not pass it to CICS.

Set the value to verify when the following conditions are true:

- To ensure that user IDs are always accompanied by a password
- When you are using an SNA product that does pass the password to CICS

Setting up a CICS region to flow user IDs

A region flows user IDs to a remote system if the `OutboundUserIds` attribute of the local CD entry for the remote system is set to `sent`, `sent_maybe_with_password`, or `sent_only_with_password`. If the remote system that you are sending to is unable to receive inbound user IDs, set the `OutboundUserIds` attribute to `not_sent`.

By default, CICS never flows passwords with these user IDs. This is because all of the local user IDs that are running transactions in a region have had their passwords checked. Therefore, set up remote systems in the SNA definitions to accept user IDs flowed by a region as `already_verified`.

See the security section in the TXSeries documentation library for more detailed information about security.

4.6 Access TXSeries CICS programs as web services

At this point, we are ready to explore how TXSeries can be accessed by several different environments (both servers and clients). Specifically, we will explore how TXSeries assets can be exposed as web services.

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. The technology is called *web services* because it integrates services (applications) using web technologies (the Internet and its standards).

These services can be new applications, or be wrapped around existing business functions to make them service-oriented architecture (SOA)-enabled. Services can rely on other services to achieve their goals.

Figure 4-28 illustrates the concept of web services.

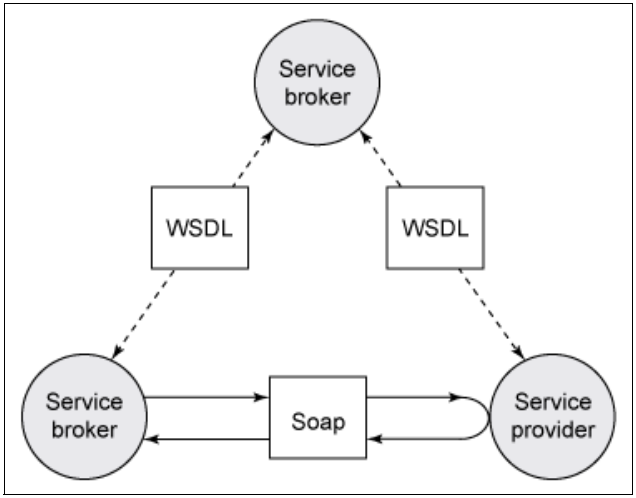


Figure 4-28 Web services

TXSeries supports web services in the following ways:

- TXSeries for Multiplatforms V8.1 and later natively supports inbound SOAP requests, wherein existing programs can be exposed as services. When these programs are deployed as services, any SOAP web service requester can access these server programs over Hypertext Transfer Protocol (HTTP). This native support uses IBM WebSphere Application Server Liberty profile, which is included with TXSeries to act as a web server.

  Requirement: TXSeries for Multiplatforms V7.1 requires you to install SupportPac CN01 to enable inbound SOAP support.

- You can expose existing TXSeries CICS programs as web services by using WebSphere Application Server, using the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) resource adapter available in CICS Transaction Gateway.

This section covers enabling TXSeries assets as web services using both the alternatives.

Choosing from these two options depends on your architecture and setup:

- The first option should suffice to enable accessing TXSeries.
- If you already have a Java EE-heavy setup running on an application server like WebSphere Application Server Network Deployment edition, the second option should suit better.

4.6.1 Accessing TXSeries web services using the native inbound SOAP support

As of version 7.1, TXSeries supports the ability to access TXSeries CICS applications as web services through a SupportPac called *TXSeries Inbound SOAP Support SupportPac*. This feature is included within the product and provided ready for immediate use from TXSeries v8.1 onwards.

Architecture of web services in TXSeries

TXSeries uses the architecture shown in Figure 4-29 to expose CICS server programs as web services.

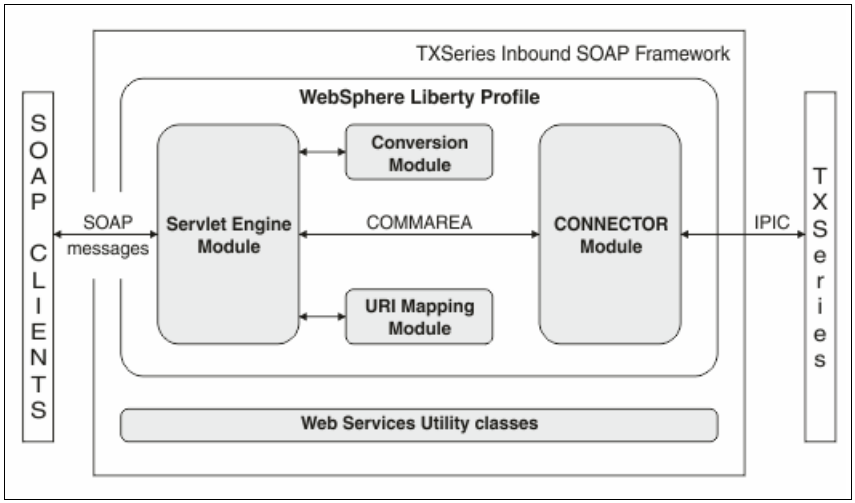


Figure 4-29 Exposing CICS as web services

WebSphere Liberty profile

The TXSeries V8.1 Administration Console uses the WebSphere Application Server Liberty profile, a highly customizable, lightweight, and dynamic runtime environment. The Liberty profile addresses the need for a minimal footprint, easy to configure, secure infrastructure for hosting web applications and web services.

Servlet engine

The Servlet engine receives SOAP requests from the SOAP client. It uses the URIMapping module, Conversion module, and Connector module to get the response, and sends back the SOAP response or SOAPFault to the SOAP client.

URIMapping module

Based on the Uniform Resource Identifier (URI) requested by the SOAP client, the URIMapping module finds the CICS program, the region it has to connect, and provide these details to the Servlet engine.

Conversion Module

The Conversion Module is a support module for the Servlet Engine, which receives the SOAP Body and converts it to the communication area (COMMAREA) byte buffer, and vice versa.

Connector Module

The Connector Module actually connects to the TXSeries and calls the CICS program with the COMMAREA byte buffer, receives back the COMMAREA, and sends it back to the Servlet Engine.

WebServices Utility Classes

The WebServices Utility Classes are DFHLS2WS utility classes, which are used for creating Web Services Description Language (WSDL) and WSBind artifacts.

Role of the wsbind file

The `wsbind` file contains the rules for creating the COMMAREA from the Extensible Markup Language (XML) and vice versa.

TXSeries support for web services

What the World Wide Web did for interactions between programs and users, web services can do for program-to-program interactions. TXSeries support for web services makes it possible for CICS applications to be integrated more rapidly, easily, and cheaply than ever before.

Application programs running in TXSeries V8.1 can participate in a heterogeneous web services environment as service providers using an HTTP transport and SOAP messages.

TXSeries includes a web services assistant utility. The web services assistant utility contains a program, `DFHLS2WS`, and a sample input file for this utility, `DFHLS2WS_SampleInput`. `DFHLS2WS` creates a new WSDL document from an existing language structure.

TXSeries provides inbound SOAP support for the following languages:

- COBOL
- PL/I
- C

To access TXSeries CICS programs as web services, take the following steps:

1. Prepare the region. You need to configure your TXSeries region so that it is able to receive web service requests.
2. Generate artifacts. Create or generate WSDL files that match with the structure of the data that the CICS programs expect as input data. The web service assistant tools can be used for this purpose.
3. Configure the WebSphere Application Server Liberty profile to make the program function as a web service.

The TXSeries program

We use a simple TXSeries CICS COBOL program, and expose that program as a SOAP-based web service. Figure 4-30 and Figure 4-31 show the CICS COBOL program, `HELLO.ccp`, and the corresponding copybook, `HELLOCOB.cpy`.

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID.   HELLO.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        01 TEXT-OUT    PIC X(15) VALUE 'Hello, World!'.

        LINKAGE SECTION.
        01 DFHCOMMAREA.
           COPY HELLOCOB.

        PROCEDURE DIVISION
            USING DFHCOMMAREA.
            DISPLAY "PROGRAM TO CHECK " TEXT-OUT.
            IF EIBCALEN EQUAL 0
               DISPLAY "COMMAREA LENGTH IS ZERO"
            ELSE
              MOVE TEXT-OUT TO DFHCOMMAREA
              DISPLAY "MODIFIED COMMAREA:" DFHCOMMAREA
            END-IF.
           EXEC CICS RETURN END-EXEC
           GOBACK.
```

Figure 4-30 COBOL program to be exposed as a SOAP-based web service

```
# cat HELLOCOB.cpy
         03 HELLOFIELD PIC X(32700).
```

Figure 4-31 COBOL copybook for the program to be exposed

Preparing the CICS region for web services

When you have identified your TXSeries CICS programs that are to be exposed as web services, you need to make the following configurations in your TXSeries regions that run the programs:

1. You need to make sure that the PD for the program is defined in the region.
2. The inbound SOAP support uses IPIC for communication between the Liberty web server and the region. Define an IPIC listener definition in the region. Figure 4-32 shows the **cicsadd** commands to define program and listener definitions. Note that `TCPAddress` must be defined as `localhost`. The `TCPService` value of `TXDEMO` is defined in the `/etc/services` file with a value of `20003/tcp`. The `services` file is in the `C:\WINDOWS\system32\drivers\etc` directory, and the name of the file is `services`.
3. Start the CICS region, as shown in Figure 4-33.

```
#cicsadd -c pd -r txdemo HELLO PathName="/var/cic
s_regions/txdemo/bin/HELLO.ibmcob" RSLKey="public
"
#
#cicsadd -c ld -r txdemo SOAPHELO TCPAddress="loc
alhost" TCPService="TXDEMO" Protocol="IPIC"
#
```

Figure 4-32 Define the program definition and listener definition

```
# cicscp -v start region txdemo
ERZ058504I/0107: Starting RPC daemon.
ERZ058502I/0101: RPC daemon is already running.
ERZ096122I/0264: Processing the 'start region' command.
ERZ096158I/0264: The region 'txdemo' is starting.
ERZ096111I/0224: Processing a start sfs_server command.
ERZ096141I/0224: Starting SFS server '/.:/cics/sfs/augus
t'.
ERZ096112I/0229: The SFS server '/.:/cics/sfs/august' is
 running.
ERZ096113I/0231: SFS server has '/.:/cics/sfs/august' su
ccessfully started.
ERZ038216I/0382: Subsystem 'txdemo' has been initialized
.
ERZ038219I/0390: Server 'txdemo' is responding to RPCs.
ERZ096126I/0276: Region 'txdemo' has successfully starte
d.
ERZ096002I/0003: The cicscp command has completed succes
sfully.
#
```

Figure 4-33 Start the region

Using the web services assistant tools

The web services assistant tools enable you to define the request and response interfaces to render the TXSeries CICS programs as service providers. These interfaces can be C language structures, or PL/I and COBOL copybooks. After the interfaces are defined, these programs can be started by using SOAP, over HTTP clients.

The tools generate the following artifacts:

The `WSBIND` file This is a file that stores the format of the structure of the expected input. The TXSeries run time uses this input to convert an incoming SOAP request into a proper structure that can be sent as COMMAREA data using IPIC to a TXSeries program.

The `WSDL` file This file is used by a SOAP client to create the requests and target them to the correct end-point.

The `DFHLS2WS` tool consists of Java archive (JAR) files that provide facilities to generate required artifacts.

You can generate artifacts using the following methods:

- Use the `DFHLS2WS` tool to generate artifacts (`WSDL` and `WSBIND` files).
- Use Java APIs to generate artifacts.

With the `DFHLS2WS` tool, generate artifacts using the following steps:

1. Run the following command, depending on your operating system:
   - On Windows systems, issue the following command:

     ```
     DFHLS2WS.bat DFHLS2WS_Input
     ```

   - On UNIX systems, issue the following command:

     ```
     DFHLS2WS.sh DFHLS2WS_Input
     ```

   In this case, `DFHLS2WS_Input` is an input file with parameters in the form of name-value pairs separated by the equal sign (=).

Figure 4-34 shows the `Input` parameter file used for the `HELLO` COBOL program.

```
# Path having high-level language data structures
PDSLIB=/var/cics_regions/txdemo/bin/

# Programming language of high-level language structure
LANG=COBOL

# File into which DFHLS2WS writes its activity log
LOGFILE=/redbook/soap_ws.log

# The level of mapping that DFHLS2WS uses
MAPPING-LEVEL=2.0

# How CICS passes data to the target program
PGMINT=COMMAREA

# The name of the CICS PROGRAM resource
PGMNAME=HELLO

# Path having high-level language structure for request
REQMEM=HELLOCOB.cpy

# Path having high-level language structure for response
RESPMEM=HELLOCOB.cpy

# Names of high-level structures in REQMEM and RESPMEM
STRUCTURE=(HELLOCOB,HELLOCOB)

# Path name of the Web service binding file
WSBIND=/redbook/hellosoap.wsbind

# Path name of file containing Web service description
WSDL=/redbook/hellosoap.wsdl

# URI that a client will use to access the Web service
URI=http://angela.in.ibm.com:9080/txws/HELLO
# Namespace for CICS to use in WSDL document
WSDL-NAMESPACE=hello
```

Figure 4-34 Input parameter file DFHLS2WS_HELLO to define a new SOAP web service for HELLO

For a detailed description of each input parameter, see the topic about input parameters in the TXSeries IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSAL2T_8.1.0/com.ibm.cics.tx.doc/reference/r_dfhls2ws.html?cp=SSAL2T_8.1.0%2F13-4-0

2. Run the `DFHLS2WS.sh` web service assistant tool, which is included with TXSeries.

3. Supply the `Input` parameter file that was just created, as shown in Figure 4-35. This generates the `WSLD` and the `WSBIND` file for the `HELLO` program.

```
# DFHLS2WS.sh ./DFHLS2WS_HELLO
DFHPI9614I Mapping level "2.0" has been requested.
DFHPI9609I Parameter "CCSID" has value "null".
DFHPI9609I Parameter "CHAR-VARYING" has value "NO".
DFHPI9609I Parameter "CONTID" has value " ".
.
.
.
DFHPI9609I Parameter "RESPMEM" has value "HELLOCOB.cpy".
DFHPI9609I Parameter "RESPONSE-CHANNEL" has value "null"
.
DFHPI9609I Parameter "RESPONSE-NAMESPACE" has value "nul
l".
DFHPI9609I Parameter "SOAPVER" has value "1.1".
DFHPI9609I Parameter "STRUCTURE" has value "(HELLOCOB,HE
LLOCOB)".
DFHPI9609I Parameter "SYNCONRETURN" has value "null".
DFHPI9609I Parameter "TRANSACTION" has value "null".
DFHPI9609I Parameter "URI" has value "http://angela.in.i
bm.com:9080/txws/hello".
DFHPI9609I Parameter "USERID" has value "null".
DFHPI9609I Parameter "WSBIND" has value "/redbook/hellos
oap.wsbind".
DFHPI9609I Parameter "WSDL" has value "/redbook/hellosoa
p.wsdl".
DFHPI9609I Parameter "WSDLCP" has value "null".
DFHPI9609I Parameter "WSDL-NAMESPACE" has value "hello".
.
.
DFHPI9587I Program "DFHLS2WS" has completed SUCCESSFULLY
.
```

Figure 4-35 Command output from the web service the WSDL and WSBIND artifacts

Alternatively, if you use the Java API to generate artifacts, the following JAR files should be in the system `CLASSPATH`.

- `common.jar`
- `common.resources.jar`
- `ecore.change.jar`
- `ecore.jar`
- `ecore.xmi.jar`
- `NamespaceContext.jar`
- `woden.jar`
- `wsdl4j.jar`
- `XmlSchema.jar`
- `xsd.jar`
- `xsd.resources.jar`
- `cwsa.jar`

In case you want to generate artifacts as part of a larger program, you can use the Java APIs to generate the artifacts. A sample program is available on the following link:

http://www.ibm.com/support/knowledgecenter/SSAL2T_8.1.0/com.ibm.cics.tx.doc/tasks/t_config_webserv_usg_jv_api.html

Configuring Liberty for web services

An application that is predeployed on Liberty profile at `$CICSPATH/wui` by TXSeries. This application (txws) is responsible to perform the following tasks:

1. Receive HTTP web service requests in the SOAP format.
2. Convert them into the correct COMMAREA structure.
3. Identify the correct TXSeries program.
4. Connect and run the identified program.
5. Send the results back to the web service (SOAP) client.

To achieve this, the application (`txws`) on Liberty profile expects the artifacts in certain directories:

1. Create the *<region name>*/*<program name>* directory in `$CICSPATH/wui/usr/ws_conf`.
2. Copy the artifacts generated with the `DFHLS2JS` tool to the following directory:

   `$CICSPATH/wui/usr/ws_conf/`*<region name>*/*<program name>*

Figure 4-36 shows the directory created for the `HELLO` program.

```
#cp /redbook/hellosoap.wsbind /usr/lpp/cics/wui/usr/ws_c
onf/txdemo/HELLO
#
#cp /redbook/hellosoap.wsdl /usr/lpp/cics/wui/usr/ws_con
f/txdemo/HELLO
#
```

Figure 4-36 Directory to copy the generated artifacts

3. Define a new web service definition. You need to use `cicsadd` to define the following attributes:

   a. `ProgramName`
   b. `RegionName`
   c. `WsBindFile`
   d. `WsdlFile`
   e. `IpicPort`

   Figure 4-37 shows the command used for the `HELLO` service that we want to create.

```
#cicsadd -c ws HELLO ProgramName="HELLO" RegionName="txd
emo" WsBindFile="hellosoap.wsbind" WsdlFile="hellosoap.w
sdl" IpicPort=20003
#
```

Figure 4-37 Define a web service definition using cicsadd

4. Restart the Liberty profile application server, and start the TXSeries region with a `StartType` value of `cold`:

   a. `$CICSPATH/wui/bin/server stop txserver`
   b. `$CICSPATH/wui/bin/server start txserver`
   c. `cicscp –v start region txdemo StartType=cold`

Testing the SOAP web service

After the Liberty profile server is started, you can check if the services are up by accessing the WSDL:

`http://<servername>:<Liberty-http-port>/HELLO?wsdl`

This should print the WSDL on your browser.

The SOAP request can be sent by creating a SOAP client using the generated WSDL. We use a tool called SOAP user interface (UI).

Figure 4-38 shows the request and response for the web services `HELLO` on the SOAP UI client.

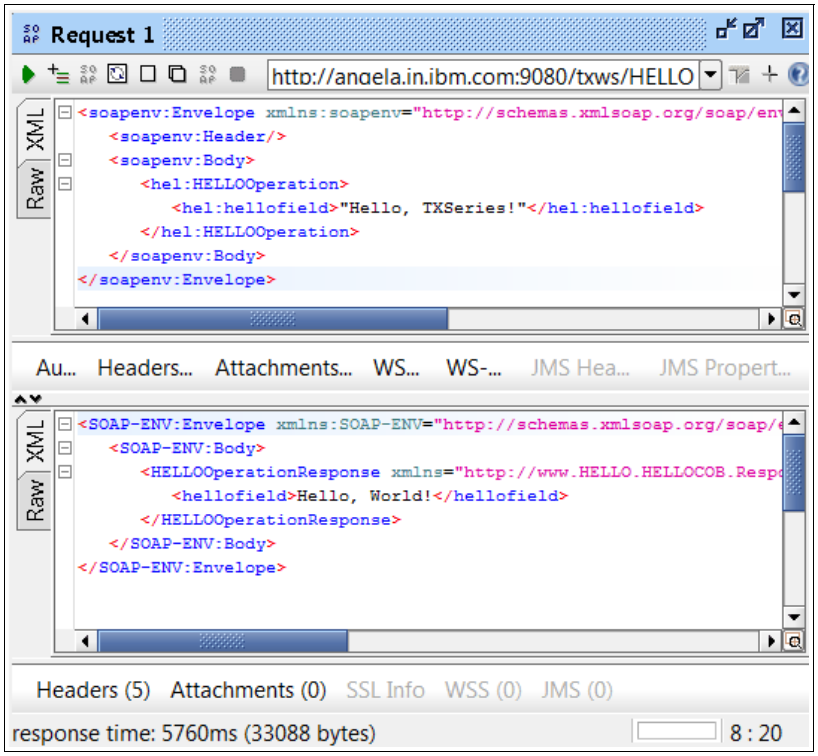


Figure 4-38 Request and response for the HELLO web service

4.6.2 Overview of IBM WebSphere Application Server

WebSphere Application Server is a comprehensive application server platform that supports the Java EE programming model and extensions. This includes Servlets, JavaServer Pages (JSP), Enterprise JavaBeans (EJB), and web services. It delivers a high-performance and extremely scalable transaction engine for dynamic e-business applications. It also delivers additional programming model enhancements to provide a secure foundation for an SOA.

The most recent version available at the time of this writing is WebSphere Application Server V8.5, supporting Java EE 6 specifications. It can easily connect to multiple back-end data sources through a standard interface with Service Data Objects (SDO). It can also be used to build dynamic web UIs with drag-and-drop development using standard JavaServer Faces (JSF).

The embedded Java Message Service 1.1 (JMS 1.1) messaging engine simplifies the task of connecting applications to form an enterprise service bus (ESB). In addition, the advanced support for JCA enables communication with various enterprise systems through a unified interface.

WebSphere Application Server V8.5 also provides a WebSphere rapid deployment feature for reducing the complexity of developing and deploying Java EE applications. All of this is provided with mixed application server support (older versions) for more flexible migration to the newest versions, and a broad platform support.

This enables increased flexibility in deployment choice, and a unified administration across disparate systems, including Linux on IBM System i®, IBM System p®, and IBM System z.

The current versions of WebSphere Application Server also integrate more closely with IBM Tivoli offerings, including embedded IBM Tivoli Access Manager. This integration enables centralized security management among Java EE and web resources. It also supports the web services security open security model for easy interoperability with third-party, client-created, or traditional security solutions.

The configuration for WebSphere Application Server V8.5 is stored in XML files. The administrator can manage the XML configuration, and easily deploy new applications components and services using web-based administration tools and through scripting.

WebSphere Application Server provides installation and administration capabilities through exposed Java Management Extensions (JMX) interface. Support for JMX enables third-party products to read and manage WebSphere software in a standard way.

Because WebSphere Application Server can run only Java code, it has to use some Java classes to access every different type of CICS region. This is also valid for TXSeries regions. Therefore, the Java application running inside the WebSphere Application Server runtime environment must include and use a set of Java classes corresponding to the different operations that we want to perform in CICS.

These Java classes are usually provided by the connectors for the specific enterprise information system (EIS) that we have to access and, specifically for CICS, by the CICS connector (CICS TG). External call interface (ECI) and external presentation interface (EPI) adapters are provided, enabling both COMMAREA and 3270-data stream-based CICS applications to interoperate effectively with WebSphere Application Server.

WebSphere Application Server uses its Java EE and JCA support productively to access CICS using CICS TG. The JCA defines standard contracts that enable bidirectional connectivity between the enterprise applications and the EISs. It also formalizes the relationships, interactions, and the packaging of the integration layer, therefore enabling enterprise application integration.

Additionally, JCA defines a common client interface (CCI) for the EIS access. The CCI defines a client API for interacting with heterogeneous EISs. It is expected that many Java EE applications will combine relational database access, using Java Database Connectivity (JDBC), with EIS access using EIS access tools based on CCI.

The connector architecture defines a standard service provider interface (SPI) for integrating the transaction, security, and connection management facilities of an application server with those of a transactional resource manager. The JDBC specification defines the relationship of JDBC to the SPI specified in the connector architecture.

For further details about JCA, see the following website:

http://docs.oracle.com/javaee/6/tutorial/doc/gipgl.html

4.6.3 Overview of CICS Transaction Gateway

This section provides a brief description of CICS Transaction Gateway and how it works. CICS TG provides the IBM implementation of the JCA for access to CICS from WebSphere Application Server. CICS Transaction Gateway V9.1 and later supports the following platforms:

- IBM z/OS
- Linux on IBM System z
- AIX
- Linux on Intel
- Microsoft Windows
- Hewlett-Packard UNIX (HP-UX)
- Oracle Solaris

In terms of functionalities, together with WebSphere Application Server, CICS Transaction Gateway can be on any system:

- On the same system where WebSphere Application Server is installed
- On an intermediate machine between the WebSphere Application Server system and the CICS system
- On the same system where CICS is

Figure 4-39 shows the CICS Transaction Gateway on the same system as the WebSphere Application Server.

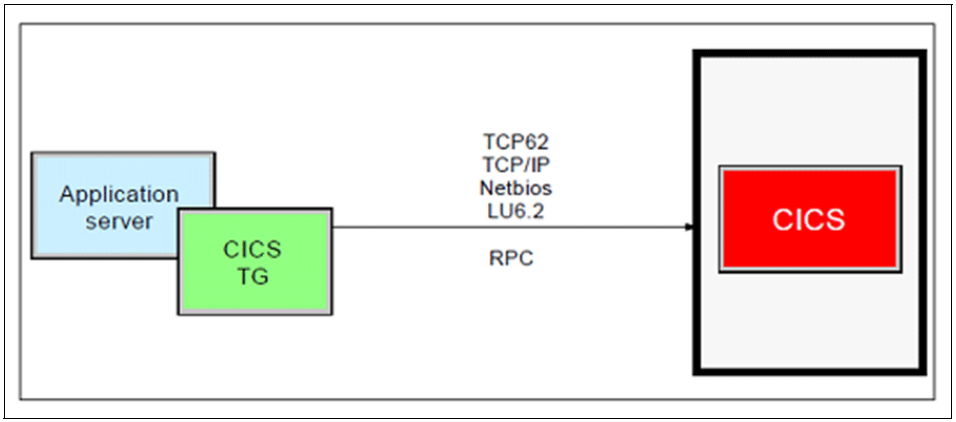


Figure 4-39 CICS Transaction Gateway on the same system as WebSphere Application Server

Figure 4-40 shows the CICS Transaction Gateway between the WebSphere Application Server and the CICS Transaction Server.

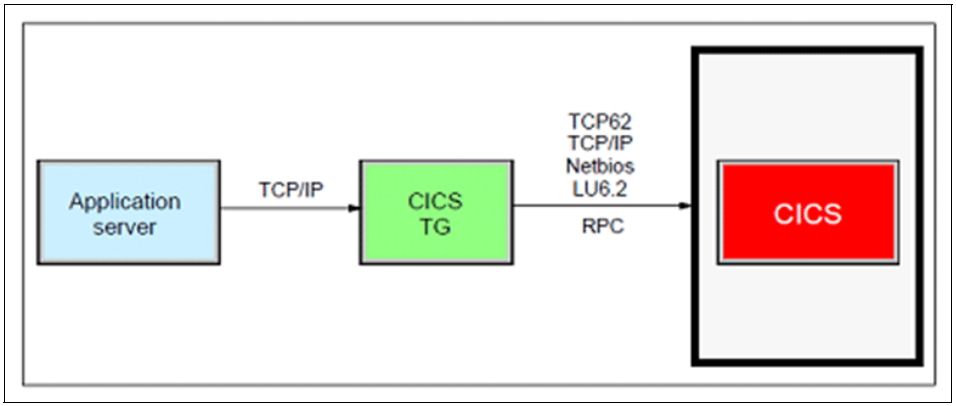


Figure 4-40 CICS Transaction Gateway between WebSphere Application Server and CICS Transaction Server

Figure 4-41 shows the CICS Transaction Gateway on the same system as the CICS Transaction Server.

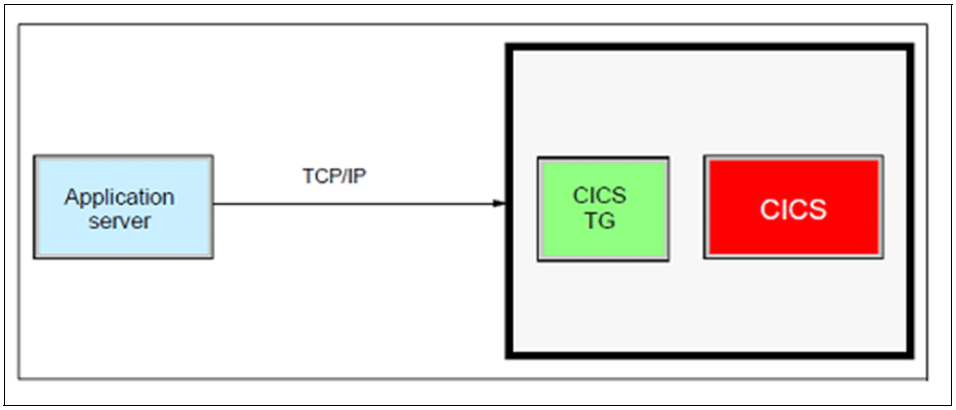


Figure 4-41 CICS Transaction Gateway on the same system as CICS Transaction Server

CICS Transaction Gateway consists of the following components:

- A *Gateway daemon* listens for incoming work and manages the threads and connections necessary to ensure good performance.
- A *Client daemon* provides the communication to CICS servers and APIs.

A Java client program can connect to the CICS TG on a distributed platform using the TCP or SSL protocols. CICS TG provides the mechanism that connects to a CICS server. In terms of protocols to use for accessing CICS, CICS TG can support all of the protocols listed in Figure 4-39 on page 178 through Figure 4-41:

- TCP62
- TCP/IP
- NetBIOS
- LU62
- RPC

The choice among them depends on both the type and version of the CICS that we have to access.

To access TXSeries regions, the protocol used by the CICS TG is always TCP/IP. TXSeries provides the support of a specific CICS resource, named CICS TCP/IP listener, for the management of the requests coming from a client application. For the WebSphere Application Server to CICS TG connection, the protocol to use depends on where the CICS TG is installed.

We can have the following two possible configurations:

- CICS TG is to be run on the same machine as WebSphere Application Server. In this case, it is more efficient to use the CICS TG classes within the WebSphere Application Server that manages the connections and the threads reducing the communication resource use. This configuration is known as *local mode*.
- CICS TG is on a different system compared to where the WebSphere Application Server runs. In this case, the protocol used to reach the Gateway daemon is TCP/IP.

For additional information about CICS Transaction Gateway, see CICS Transaction Gateway-specific documentation on the following website:

http://www.ibm.com/software/htp/cics/ctg/library/

4.6.4 WebSphere Application Server and CICS TG configuration

This book does not cover the installation steps that you must perform to have both the WebSphere Application Server and the CICS Transaction Gateway configured and running. For details about this, see their specific documentation.

In this section, we explain the configuration that we adopted for this demonstration environment.

Configuring CICS Transaction Gateway

To configure the CICS Transaction Gateway, perform the following steps:

1. Log in as root user (because we are using an AIX system).
2. Change the directory to the bin directory of this CICS Transaction Gateway installation.

   ```
   cd /opt/IBM/cicsctg/bin
   ```

3. Start the CICS Transaction Gateway configuration tool:

   ```
   ./ctgcfg
   ```

4. The Configuration Tool wizard opens, as shown in Figure 4-42.

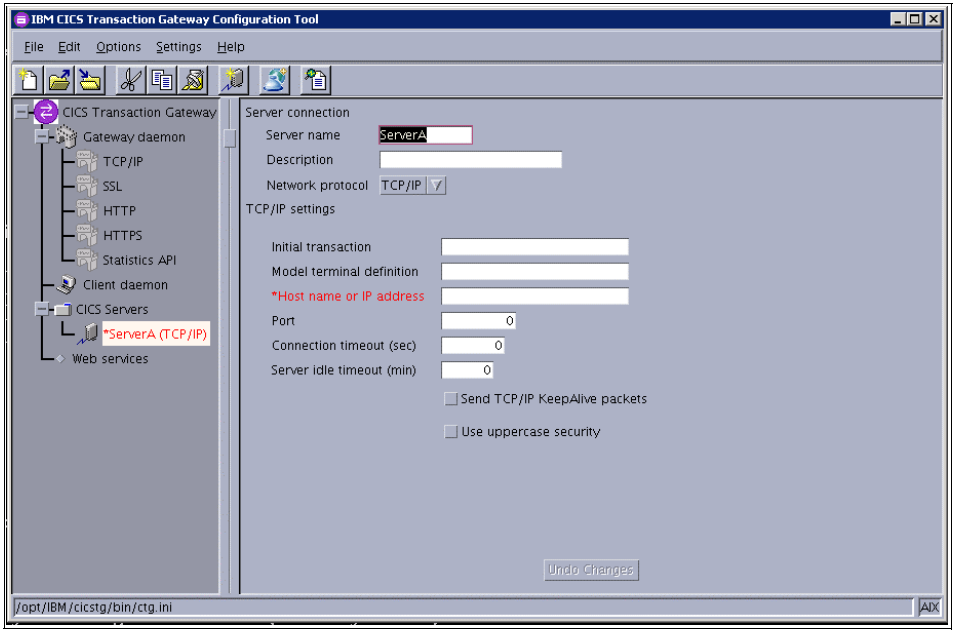


Figure 4-42 Configuration wizard

5. Type the name of the TXSeries CICS region that you must access, as shown in Figure 4-43.
6. In this case, for simplicity, we decide to use the same name with which we created the TXSeries region, but you can provide any name here. The link to the region is made using the protocol, IP address, and port number used by the CICS TCP/IP listener to receive the requests.

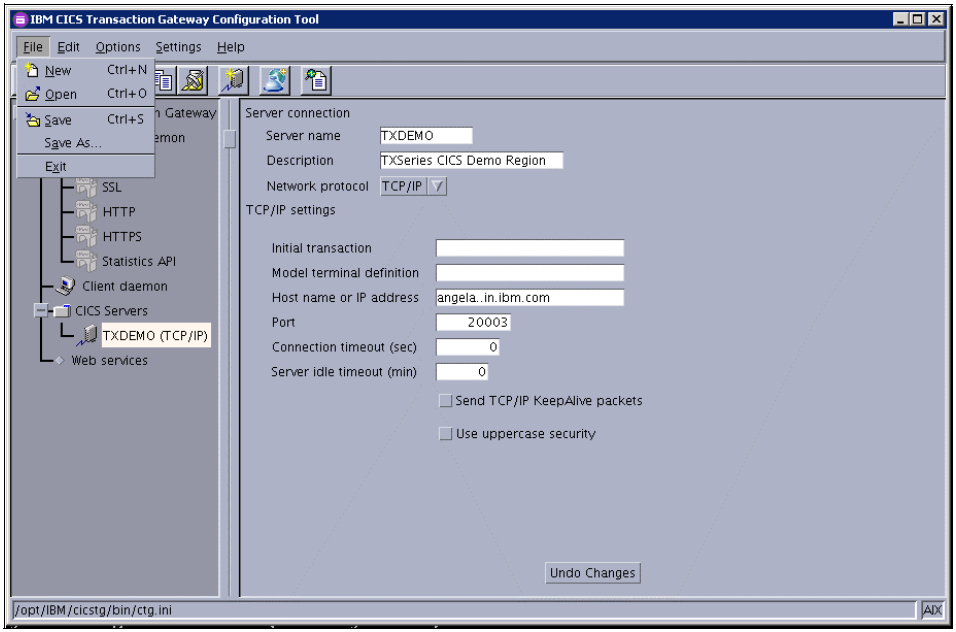


Figure 4-43 Creating a CICS Server definition

7. Click **File** → **Save,** and save the file as `ctg.ini`. See Figure 4-44.

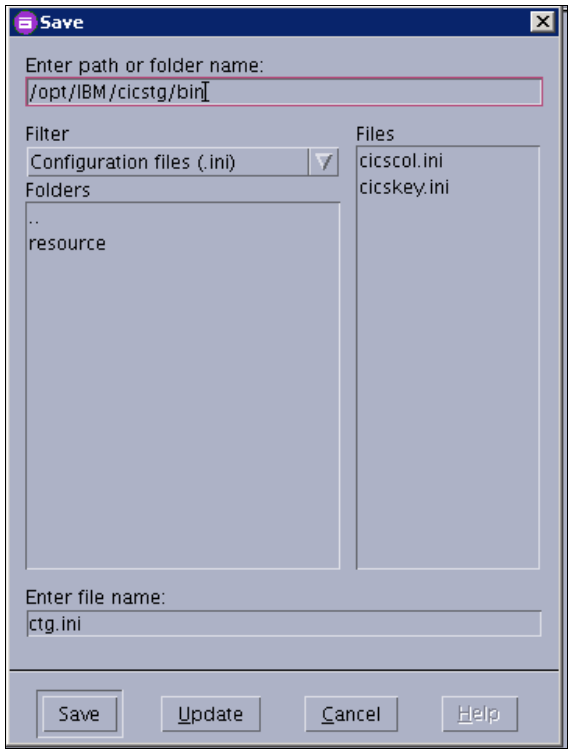


Figure 4-44 Save configuration

8. In the new window (Error: Reference source not found), click **Finish**.

The IBM CICS Transaction Gateway Configuration Tool opens with all of the entries that you created, as shown in Figure 4-45.

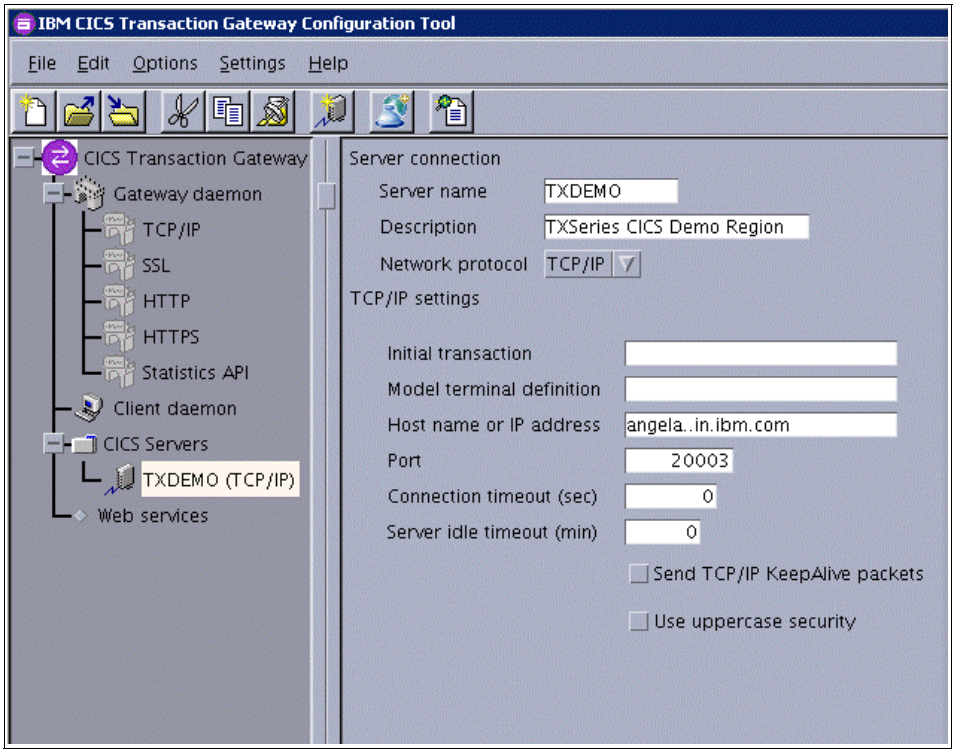


Figure 4-45 CICS Transaction Gateway configuration tool

9. Before closing the graphical interface, you have to save the configuration so that the `ctg.ini` version file is created in the CICS Transaction Gateway directory. In the Save configuration window (Figure 4-46), click **Yes**.

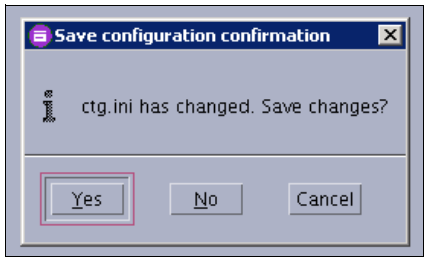


Figure 4-46 Saving the configuration

10. Start the CICS Transaction Gateway Client daemon for the TXDEMO TXSeries region. Log in as root and run the following command:

```
cicscli –S=TXDEMO
```

You see the message shown in Figure 4-47.

```
# cicscli -S=TXDEMO
CCL8001I CICSCLI - CICS Transaction Gateway Client Daemo
n Control Program
CCL0002I (C) Copyright IBM Corporation 1994, 2014.  All
rights reserved.
CCL8038I The Client daemon is starting
CCL8037I Connection to server 'TXDEMO' is being started
#
```

Figure 4-47 Messages

> **Tip:** At this point, if you get an error message informing you that the server you have just added is UNKNOWN, fix the problem by closing the IBM CICS Universal Client using the **`cicscli -X`** command.

11. Check that the connection is really available by using the **cicscli -L** command, as shown in Figure 4-48.

```
# cicscli -L
CCL8001I CICSCLI - CICS Transaction Gateway Client Daemo
n Control Program
CCL0002I (C) Copyright IBM Corporation 1994, 2014.  All
rights reserved.
CCL8041I The Client daemon is using the following server
s:
CCL8042I Server 'TXDEMO' (using 'TCPIP' to 'angela.in.ib
m.com') is available
#
```

Figure 4-48 Command to check connection

12. Because the connection is established, try to connect to the TXSeries CICS region txdemo.

> **Remember:** The TXSeries region name is txdemo (lowercase), and the name with which the CICS Transaction Gateway Client daemon knows this region is TXDEMO (uppercase). We choose these two different names to demonstrate that the name for the CICS Transaction Gateway Server Name definition in `ctg.ini` file can be any case.

To connect to TXSeries, use the following command:

```
cicsterm -S=TXDEMO
```

13. Type the CEMT transaction, which is shown in Figure 4-49.

> **Restriction:** The minimum panel size must be of 25 rows by 80 columns. If it is less than this size, the `cicsterm` emulator ends.

```
I TASK
STATUS:  RESULTS - OVERTYPE TO MODIFY

  Tas(        23) Tcl(DFHTCL00)  Tra(CEMT) Fac(EOMN) Run  Inf Ter
         User(CICSUSER) Processid(   5505302)
         Currentprog(DFHCEMT )

                                                          APPLID=txdemo
RESPONSE:  NORMAL
PF 1 HELP       3 END                    7 SBH 8 SFH 9 MSG 10 SB 11 SF
1B EOMN                  TXDEMO                                   01/009
```

Figure 4-49 CEMT transaction

14. Run the **CESF** transaction to log off from CICS, and the **EXIT** command to close the cicsterm emulator.

4.6.5 Configuring WebSphere Application Server

To configure WebSphere Application Server, complete the following steps:

1. Install the resource adapter.
2. Create the connection factories.
3. Deploy the demonstration application.

The CICS Transaction Gateway provides two JCA resource adapters, `cicseci.rar` and `cicsepi.rar`. They are available in the deployable directory of the CICS Transaction Gateway installation path. For this demonstration, we used the `cicseci.rar` resource adapter because we had to perform an ECI call and therefore pass a COMMAREA on to the TXSeries program.

To install the resource adapter, perform the following steps:

1. Open the WebSphere Application Server administrative console and log in. In this case, we use the following Uniform Resource Locator (URL) in the web browser:

   `http://9.100.194.80:9060/admin`

   > **Remember:** The minimum panel size must be of 25 rows by 80 columns. If it is less than this size, the `cicsterm` emulator stops.

2. In the administrative console Welcome pane, select **Resources** → **Resource Adapters**. You have to create the JCA Resource Adapter for connecting to the TXSeries CICS region txdemo.
3. In the Preferences pane (Figure 4-50), click **Install RAR**.

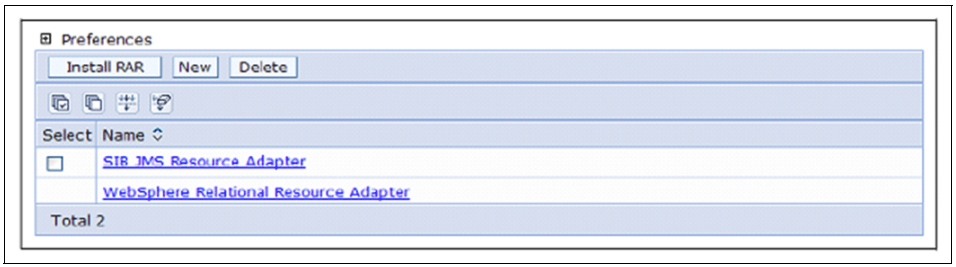


Figure 4-50 Preferences pane

4. In the Install RAR file pane, select **Local Path**. Click **Browse** to locate the resource adapter `cicseci.rar` file, as shown in Figure 4-51. Click **Next**.

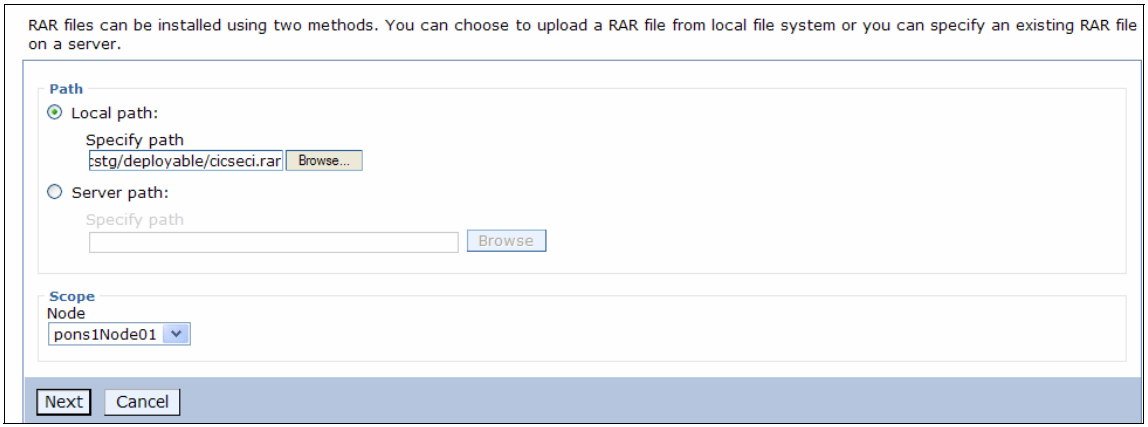


Figure 4-51 Installing the RAR file

5. The Resource Adapter pane opens. Click **OK**.

> **Information:** We are working with the WebSphere Application Server version named Network Deployment, and so with a WebSphere Application Server Cluster environment of two nodes. Therefore, we have to repeat the CICS Transaction Gateway configuration on both the nodes in our WebSphere Application Server cell. The scope for our Resource Adapter definition is at the Node level.

The ECIResourceAdapter is now added to the list of installed adapters, as shown in Figure 4-52.

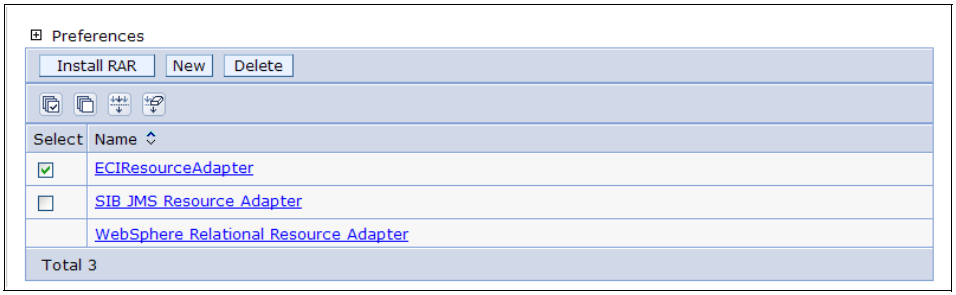


Figure 4-52 List of installed adapters showing ECIResourceAdapter

6. To add the changes to the master repository, click **Save**. Click **Save** again.

Important: For our demonstration, we are working in a cluster environment. Therefore, every time we save we also have to remember to select the **Synchronize changes with Nodes** check box.

7. Now you have to create the connection factories. Each installed resource adapter should have one or more connection factories associated with it. The connection factory is used to define the connection to the target CICS system. As such, it has to contain the following information:
   - The CICS region that is the target of the request through this connection factor.
   - The connection details of the Gateway daemon that are used. In addition, the connection factory can contain other optional information:
     - The security credentials that are used for communication
     - The mirror transaction name that should be used

Remember: As already mentioned in one of the previous sections, if WebSphere Application Server and CICS Transaction Gateway are installed on the same system, the most efficient configuration for them is in local mode. This means that the Gateway daemon is not necessary, and it does not have to be started. Every request coming from the WebSphere Application Server is sent directly to the Client daemon.

Options: At least one connection factory is required to use a resource adapter. However, it is also possible to create multiple connection factories, each with different options (such as differing CICS region names). You can use these different connection factories to enable different applications to use connections with different properties, or to different CICS regions.

To create connection factories, perform the following steps:

1. From the Resource Adapters pane, click **ECIResourceAdapter**.

   The Configuration pane opens listing the General Properties and the Additional Properties of the resource adapter, as shown in Figure 4-53 on page 190. Usually, you do not have to change the general properties of the resource adapter.

2. Click the **J2C connection factories** link shown in Figure 4-53.

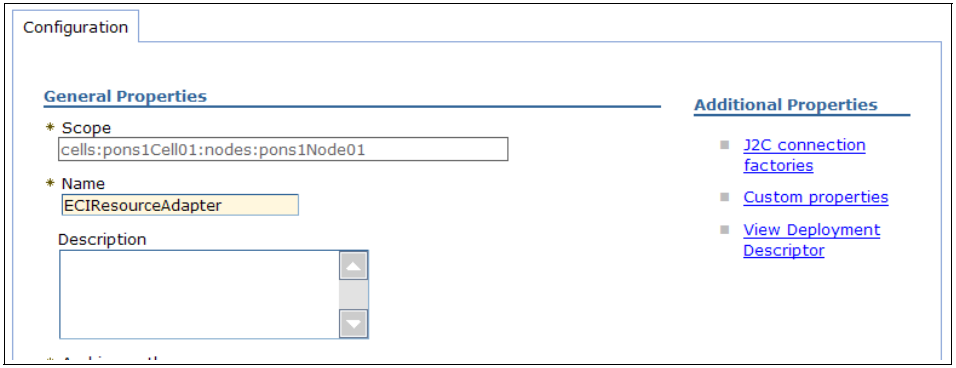


Figure 4-53 Configuration pane

3. In the new window that opens, click **New**.
4. The General Properties pane of the connection factory to be created opens, as shown in Figure 4-54:
   a. In the Name field, enter the value `CICSECI_CF`.
   b. In the Java Naming and Directory Interface (JNDI) name field, enter `eis/cicseci` for this connection factory.
   c. Click **Apply**.
   d. From the left of the window, click the **Custom properties** link.

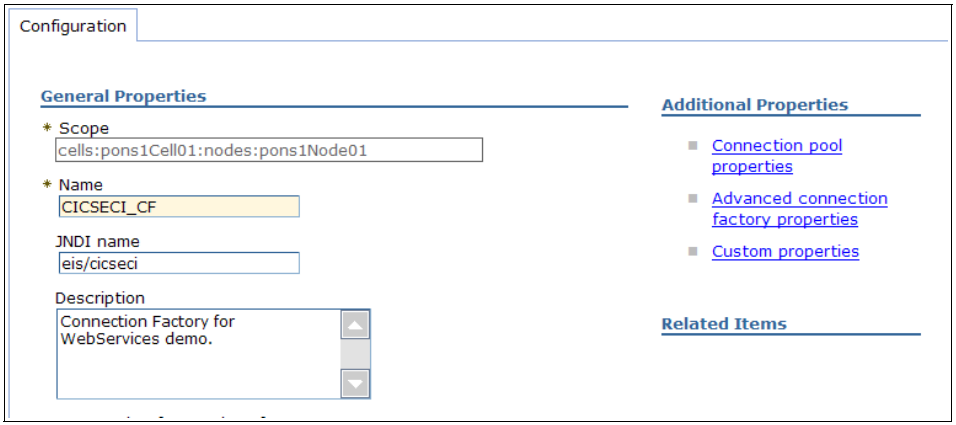
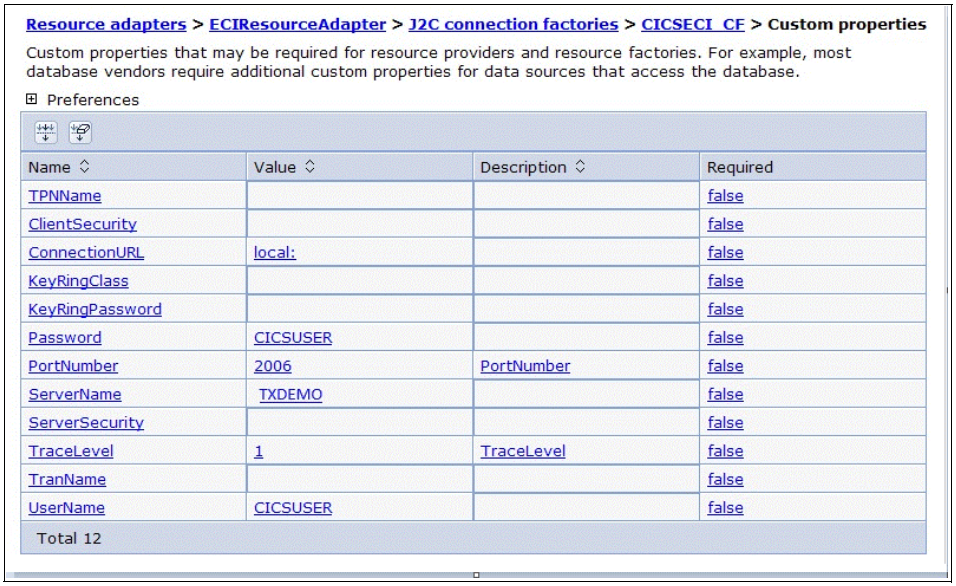


Figure 4-54 Setting general properties for JCA connection factories

5. The pane shown in Figure 4-55 opens.

   The client properties are the parameters used by the CICS ECI resource adapter to communicate with a Gateway daemon (or in local mode) and a target CICS region. For this demonstration, set the values shown in Figure 4-55 and save the configuration.



Resource adapters > ECIResourceAdapter > J2C connection factories > CICSECI_CF > Custom properties

Custom properties that may be required for resource providers and resource factories. For example, most database vendors require additional custom properties for data sources that access the database.

Preferences

Name	Value	Description	Required
TPNName			false
ClientSecurity			false
ConnectionURL	local:		false
KeyRingClass			false
KeyRingPassword			false
Password	CICSUSER		false
PortNumber	2006	PortNumber	false
ServerName	TXDEMO		false
ServerSecurity			false
TraceLevel	1	TraceLevel	false
TranName			false
UserName	CICSUSER		false

Total 12

Figure 4-55 Custom properties

The last step is to deploy the application:

1. Use the administrative console to log in.
2. In the administrative console Welcome pane, click **Applications** → **Install New Application**.

Information: The following factors apply to the parameter properties:

- **`ConnectionURL`** is the URL defining the TCP/IP host name and protocol used to communicate with the Gateway daemon. For the reason already explained, we use `local:` as the value for this field. In case you are using the Gateway daemon, you can use a URL in one of the following formats:
  - `tcp://9.100.194.80`
  - `tcp://pons1.mop.ibm.com`
- **`PortNumber`** is the TCP/IP port number that the Gateway must be listening on. By default, it is `2006`. In our case, it is insignificant.
- **`ServerName`** is the APPLID of the CICS region that requests are sent to. We use `TXDEMO` (uppercase) because it has to match with the name used in the `ctg.ini` configuration file to identify the TXSeries CICS region that we have to access.
- **UserName** is the user ID to be flowed to the Gateway and onto CICS. In our case, we use `CICSUSER`.
- **Password** is the password to be flowed to the Gateway. We set it to `CICSUSER`.

For all of the other parameters, use their default values.

3. The pane shown in Figure 4-56 opens. Enter the location of the Enterprise Archive (EAR) file, `CICSWASTXEAR.ear`. Click **Next**.

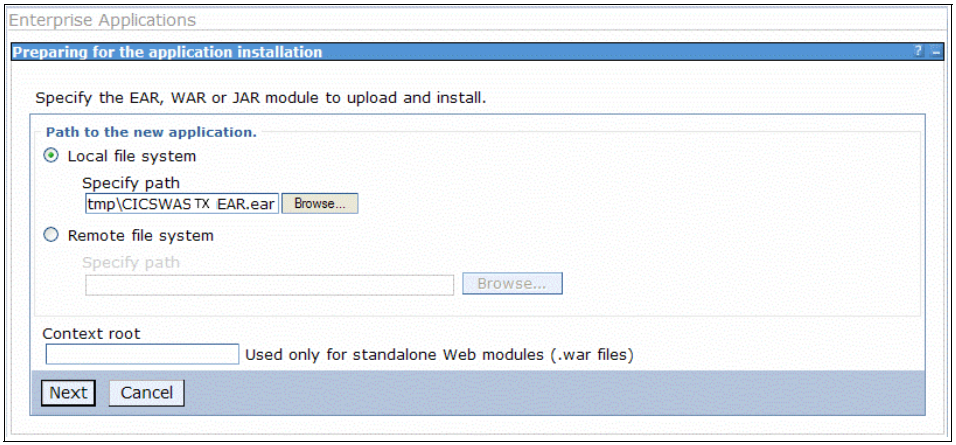


Figure 4-56 Installing new applications

4. You are prompted with the details of the six-step process for installing new applications. As in Step 1 on page 191, select the installation options, as shown in Figure 4-57. Click **Next**.

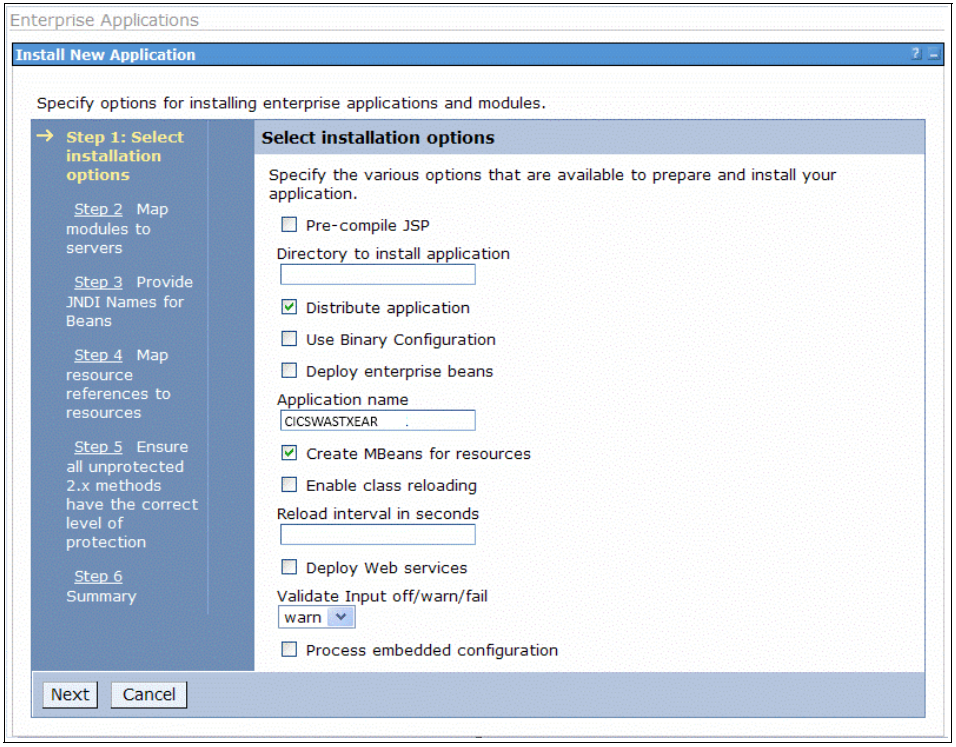


Figure 4-57 Specifying the options

5. As in Step 2 on page 191, highlight the WebSphere Cluster. In our case, **Test1** and the two web servers that we want to use for this demonstration.

6. Select the module included in the `.ear` file and click **Apply**, as shown in Figure 4-58.

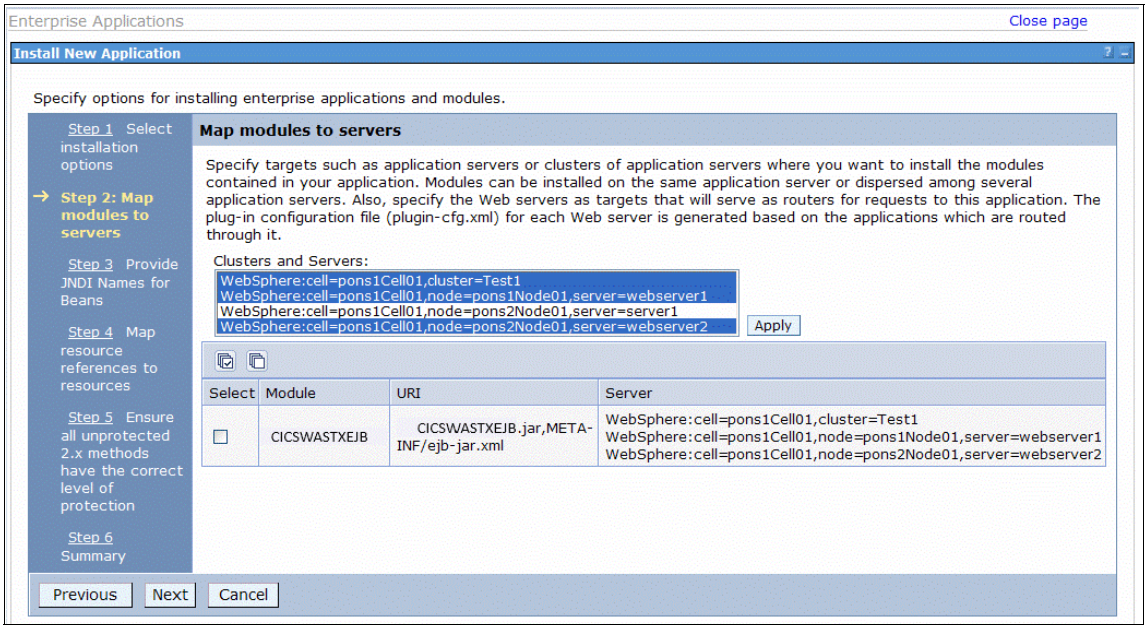


Figure 4-58 Mapping the modules to servers

7. As in Step 3 on page 192, click **Next**.
8. As in Step 4 on page 193, perform the following steps:
   a. Select the box relating to the module at the bottom of the window.
   b. Select the existing JNDI name `eis/cicseci` (this is the same name that you used during the connection factory creation).
   c. Click **Apply**.
   d. Click **Next**.
9. In the Application Resource Warnings window, click **Continue**.
10. As in 5 on page 193, click **Next**.
11. Click **Finish** to exit the six-step process. Confirm that the deployment was successful, and save your changes to the WebSphere repository, synchronizing the nodes.

12. The installed EAR (`CICSWASTXEAR`) file is now displayed in the list of the enterprise applications, as shown in Figure 4-59. Select the check box next to `CICSWASTXEAR` and click **Start** to start it properly.

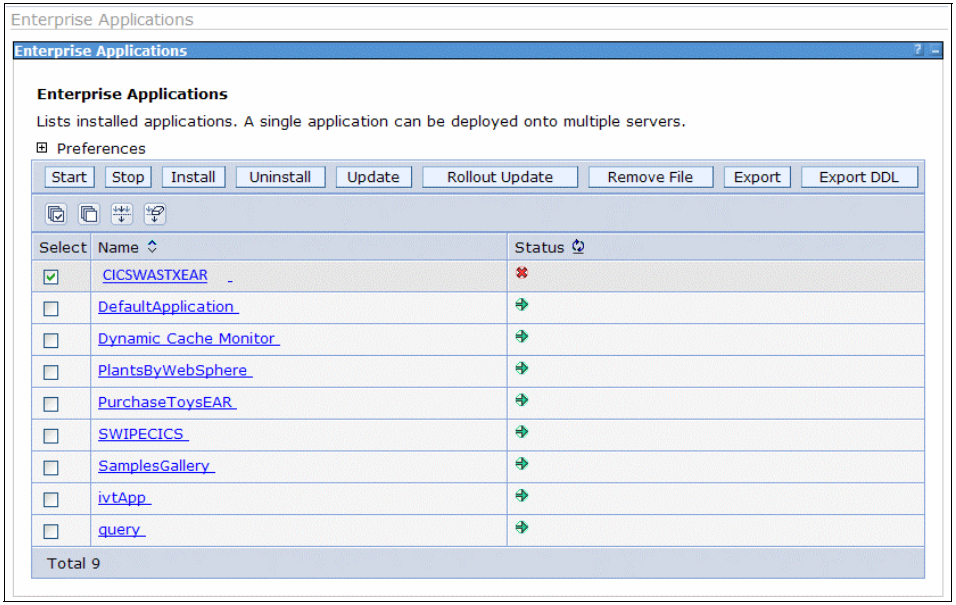


Figure 4-59 List of installed applications

In WebSphere Application Server, you can choose and run all kinds of Java applications. As you can see from Figure 4-59, the example application is made by a business logic part only (`.jar` file). It is also possible to have the presentation part and the business logic part together in the same EAR file (this is what the Java EE specification encourages the user to do).

Alternatively, you can have only a presentation part using the resource adapter and communicating with TXSeries (this means you must not have EJB components, but have JSPs and servlets only). The same observations are valid for a web services Java application running as business logic in WebSphere Application Server.

In the following sections, we explain the web services scenario that we adopted for this demonstration environment, and how to develop the corresponding Java code.

Important: Verify that the JNDI name associated with the module is now `eis/cicseci` (with no EJB components, but JSPs and servlets only). The same observations are valid for a web services Java application running as business logic in WebSphere Application Server.

4.6.6 Accessing TXSeries server programs as web services

This section describes how users of TXSeries can access the existing CICS applications as web services. This requires the reader to possess some knowledge of web services and the IBM WebSphere Studio Application Developer Integration Edition product. We assume that WebSphere Studio Application Developer Integration Edition is already installed and configured.

> **Remember:** The scenario in this section uses WebSphere Studio Application Developer Integration Edition to create an application using TXSeries as web services. If you have to use IBM Rational Application Developer, you must still follow steps equivalent to the ones described in this section.

Therefore, you need to explore various ways of accessing TXSeries CICS applications as web services using tools and products external to TXSeries. The solution we outline in this section, and the solution we chose for the implementation of this demonstration environment, is based on how to enable users to access the existing COMMAREA-based TXSeries CICS server programs as services, without any modification to the server programs.

Figure 4-60 illustrates how TXSeries server programs can be accessed as web services. Using the CICS ECI resource adapters included with WebSphere Studio Application Developer Integration Edition and the TXSeries server programs, the web service wizards of WebSphere Studio Application Developer Integration Edition generate the Web Services Description Language (`.wsdl`) files corresponding to the TXSeries server programs.

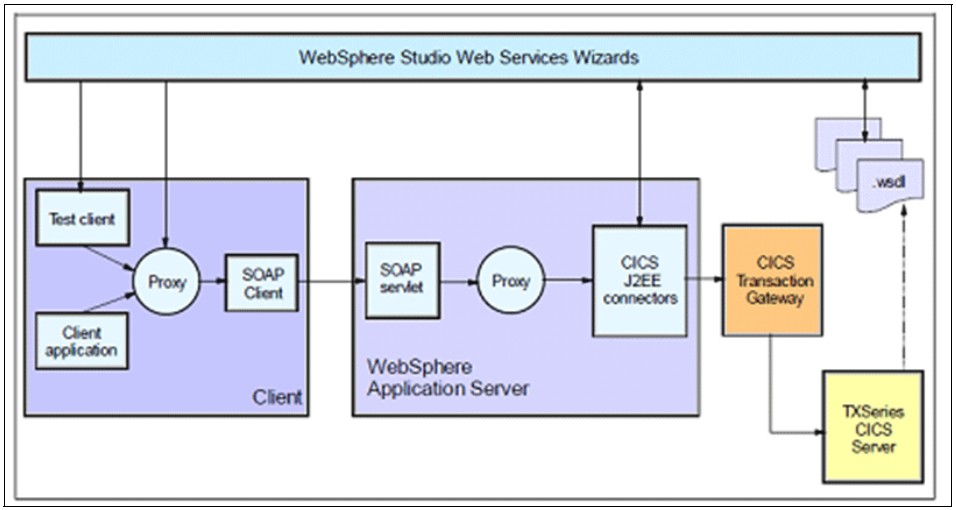


Figure 4-60 WebSphere Studio Web Services Wizards

Important: In this example, the TXSeries server programs are COMMAREA-based, but 3270-based programs can also be used.

You can deploy the generated web service files and the Java EE connectors of CICS in the WebSphere Application Server environment. The application server provides a SOAP engine that accepts the incoming SOAP requests, and routes them to the appropriate application.

In this case, the request is routed to the TXSeries servers through the CICS Transaction Gateway. You have to configure CICS TG to communicate with the appropriate CICS region where the server programs are. For more information, see 4.6.4, “WebSphere Application Server and CICS TG configuration” on page 180.

Remember: Any change to the server programs requires the regeneration or modification of the `.wsdl` files, and the redeployment of the enterprise application.

WebSphere Studio Application Developer Integration Edition also provides a WebSphere Application Server test environment. Use this environment to test the generated web services before deploying them on the production system. WebSphere Studio Application Developer Integration Edition also includes tools that generate test clients for web services. This helps you test the server-side artifacts before developing client applications.

Developing new services for existing TXSeries server programs

In this section, we describe how to develop enterprise services for TXSeries CICS ECI-based (COMMAREA-based) programs. An enterprise service is a standard way of modeling large-scale application assets. It is not just any web service. It is much larger in scale and complexity, often accessing various EIS systems, such as TXSeries.

WebSphere Studio Application Developer Integration Edition provides tools to create enterprise services. To support the generation of the ECI-based enterprise services that we require, we use WebSphere Studio Application Developer Integration Edition version 5.1.1 or later.

You can generate the ECI services using both COBOL and C server-side programs (in our case, we use a COBOL program). You can deploy enterprise services and business processes developed using WebSphere Studio Application Developer Integration Edition on IBM WebSphere Application Server.

TXSeries CICS ECI services

Using the CICS ECI resource adapter that is included with the WebSphere Studio tools, you can build and test TXSeries CICS ECI services that access CICS transactions on servers. Complete the following steps to generate the CICS ECI services:

1. Import the CICS ECI adapter (JCA adapter for ECI-based CICS programs).
2. Create a service project. The service project contains the files that compose each enterprise service or process.
3. Import the TXSeries CICS server program, which is the COBOL (or C) file that is used to create the service interface's input and output messages.
4. Generate an enterprise service for the CICS ECI resource adapter. This task generates three WSDL files that define the enterprise service for the CICS ECI resource adapter.

At the end of these steps, three WSDL files containing definitions for the service are generated, and we go on to generate the deployment code for the service. All these steps are performed in the Business Integration perspective of WebSphere Studio Application Developer Integration Edition. This perspective configures the layout of the workbench to facilitate the development of JCA services.

Importing the CICS ECI adapter

To generate enterprise services for a CICS JCA resource adapter, you must have the resource adapter imported into the workbench. To import the resource adapter, perform the following steps:

1. If you are not working in any wizard, you can start the Connector Import wizard by performing the following steps:

   a. From the menu bar, select **File** → **Import**. The Import wizard opens.
   b. In the Import wizard, select **RAR file**. The Connector Import wizard opens.

   If you are in the New Service wizard, click the **Import resource adapter** button to start the Connector Import wizard. You can see this button in the New Service wizard (Figure 4-64 on page 203).

2. In the Connector Import window (Figure 4-61 on page 199), perform the following steps:

   a. Click **Browse** to find the resource adapter (`.rar`) file.

      You can find the CICS JCA resource adapters included with the WebSphere Studio product in the following directory:

      `<WS_installdir>\resource adapters\ctg510`

      Here `<WS_installdir>` is the directory where WebSphere Studio is installed. In this case, choose the `cicseci.rar` file.

b. Select the Connector project into which you want to import the ECI resource adapter.

c. The **Standalone connector project** check box is selected by default, as shown in Figure 4-61. You can import the adapter into a stand-alone connector project, or into a connector project in an Enterprise Application project.

d. Click **Finish** to complete the import.

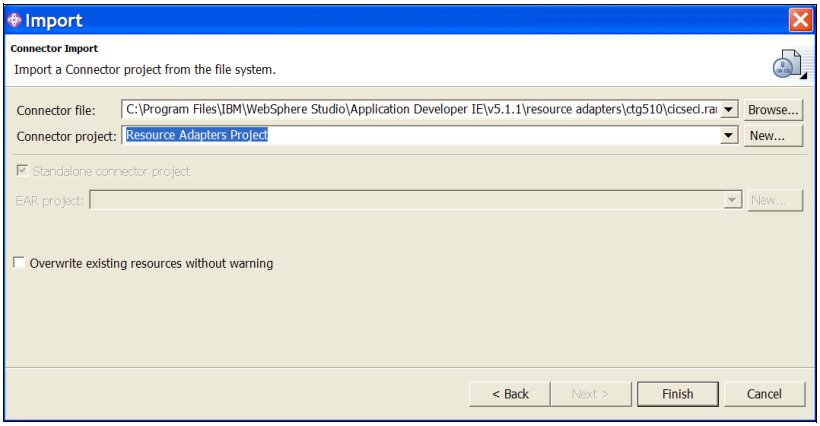


Figure 4-61 Importing the resource adapter

3. Select the **Java EE Hierarchy** view and expand **Connector Modules** to see the new connector project that contains the imported resource adapter.

Creating a service project

The service project contains the files comprising each enterprise service or process. Complete the following steps to start and use the New Service Project wizard to create the service project:

1. In the Business Integration perspective, from the workbench menu bar, select **Window** → **Open Perspective** → **Business Integration**.
2. From the workbench menu bar, select **File** → **New** → **Service Project**.

3. The New project window opens, as shown in Figure 4-62. There, perform the following steps:
   a. In the Project name field, type the name of your new project (in this case, `CICSWASTX`). Do not use spaces or special characters in the project name.
   b. You can either use the default location or click **Browse** to specify a different directory where to store the new project.
   c. Click **Next**.

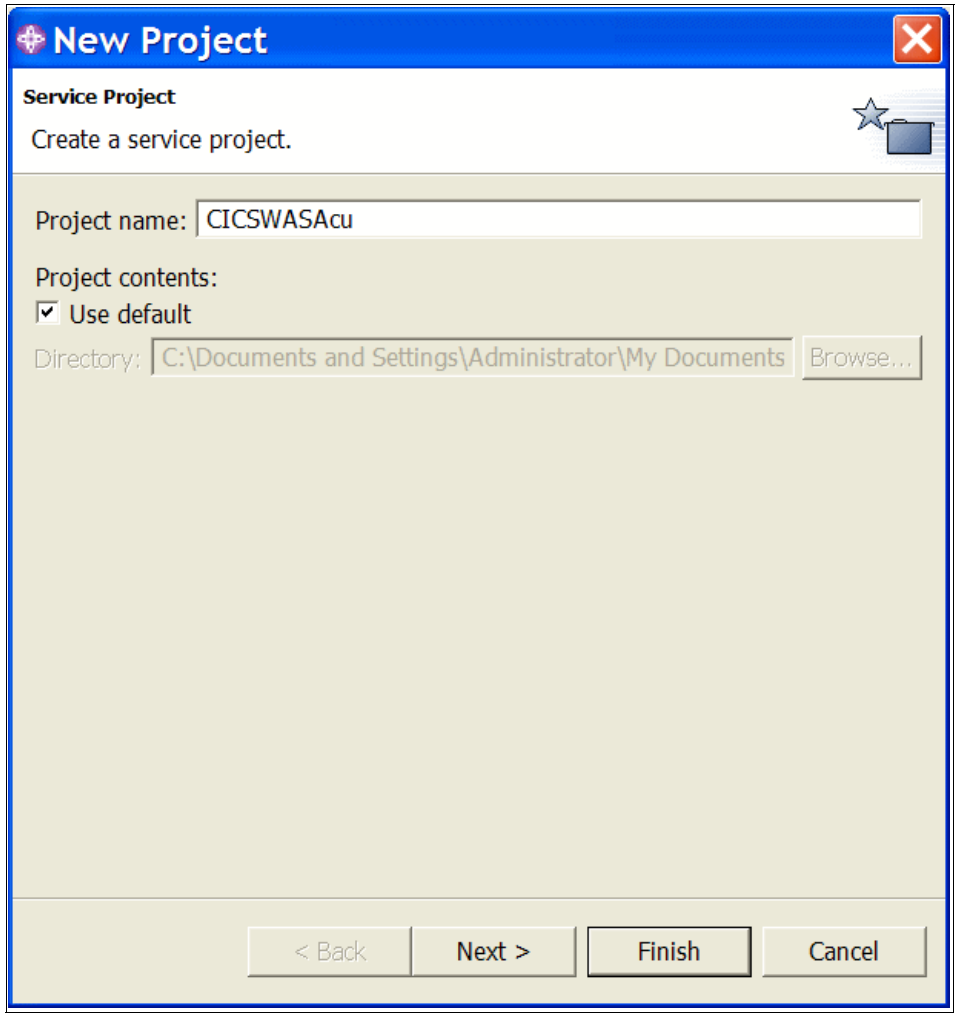


Figure 4-62 New Project

4. On the Java Settings page, complete the following steps:
   a. Click the **Source** tab. The source folder is, by default, based on the location of the file to be imported.
   b. Optional: Click the **Projects** tab to select the projects to be included on the build path.
   c. Optional: Click the **Libraries** tab to add JAR files and class folders to the build path.
   d. Optional: Click the **Order** tab to change the order of the build path settings as required.
   e. Optional: If you want to change the location of the build output, modify the default output folder field.
   f. Click **Finish**.

In the Services view, you see the new service project listed under Service Projects.

Importing the TXSeries CICS server program

The TXSeries COBOL server program is used to create the service interface's input and output messages. To import it, perform the following steps:

1. Select the project that you have just created by selecting **File Import**.
2. On the Import wizard page, select **File system**. Click **Next**.
3. In the Import window (Figure 4-63 on page 202), perform the following steps:
   a. In the Directory field, browse to the directory where the file (`CUSTECI.ccp`) that you want to import is, and click **OK**.
   b. Select the folder in the pane on the left.
   c. In the pane on the right, select the check box next to the file that you want to import. Import the following resource:
      - COBOL copybook file (`.ccp`, `.cbl`, `.cpy`, or `.cob` files)
      - C file (`.ccs`, `.h`, or `.c` files)
   d. Ensure that the project and package directory you want to use is in the Into Folder field. You can browse to select a different project and package, if required. Packages that do not exist are created for you.
   e. In the Options field, you can select **Overwrite existing resources without warning** and **Create complete folder structure**, if necessary. In this case, we select **Create selected folders only**.

4. Click **Finish**, as shown in Figure 4-63.

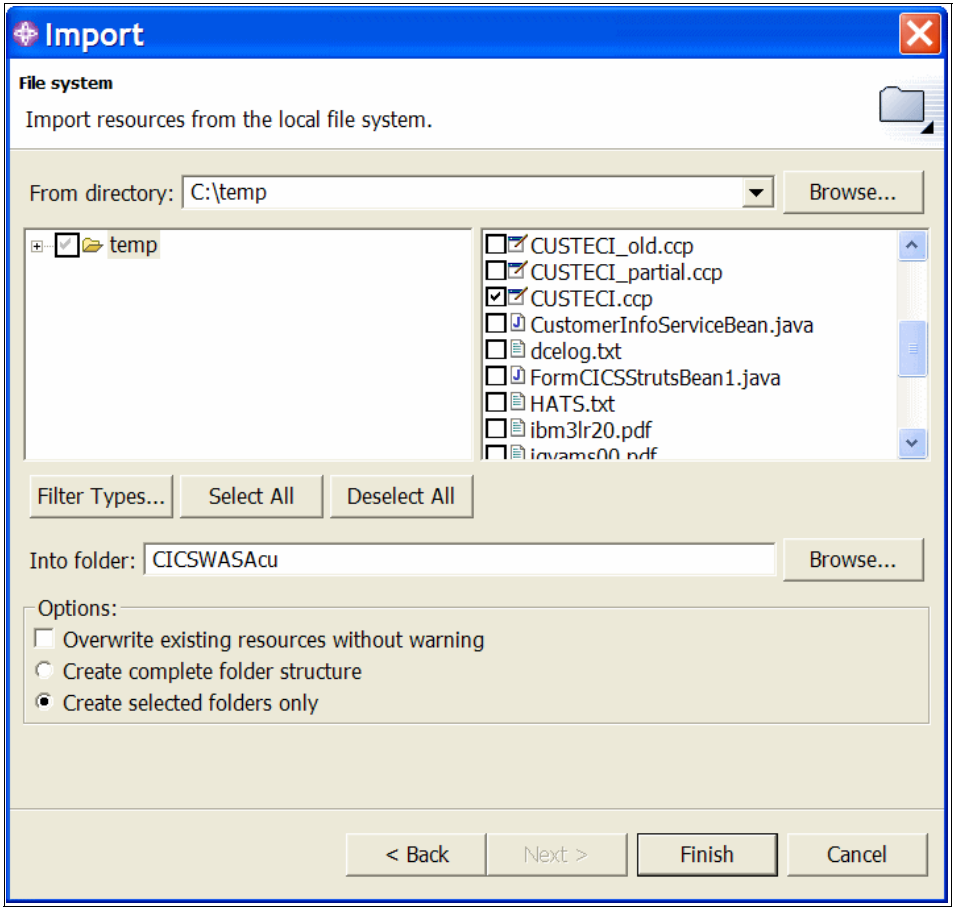


Figure 4-63 Importing the resources

The file is imported to the directory that is specified. You can see it in the Services view.

Generating an enterprise service for the CICS ECI resource adapter

This task generates three WSDL files that define the enterprise service for the CICS ECI resource adapter. To generate this service, perform two steps:

1. Use the Create Service wizard to generate the service WSDL files.

   Note: At the end of this step, only the service and port definitions are done.

2. Use the WSDL editor to add operations and bindings to the CICS ECI service.

Perform the following steps to generate the WSDL files (without any operations, messages, or binding (implementation) information) for the CICS ECI service:

1. Select the service project in which you want to store the generated enterprise service. Right-click and select **New Service built from**.
2. The Create Service wizard opens (Figure 4-64). Select the **CICS ECI** resource adapter and click **Next**.

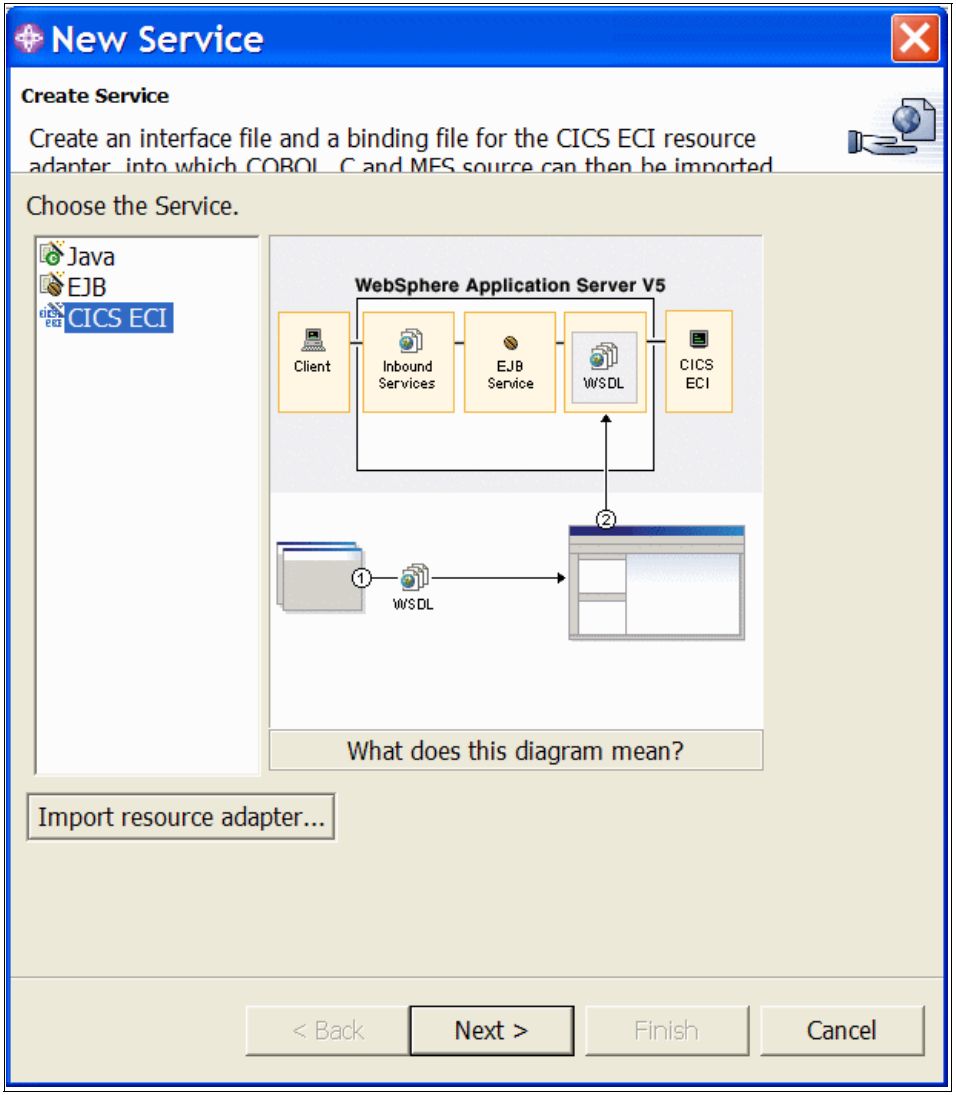


Figure 4-64 Choosing a service

Important: If you do not see the CICS ECI resource adapter, you have to click the **Import resource adapter** button and follow the instructions provided in “Importing the CICS ECI adapter” on page 198.

3. Specify the CICS connection information. In the Connection Properties page (Figure 4-65), enter the CICS Transaction Gateway address, the CICS server name, the User name, Password, and any other required properties that you want to use. If the service is used in a process, you have to specify the connection factory’s JNDI name in the **JNDI Lookup name** field. Click **Next**.

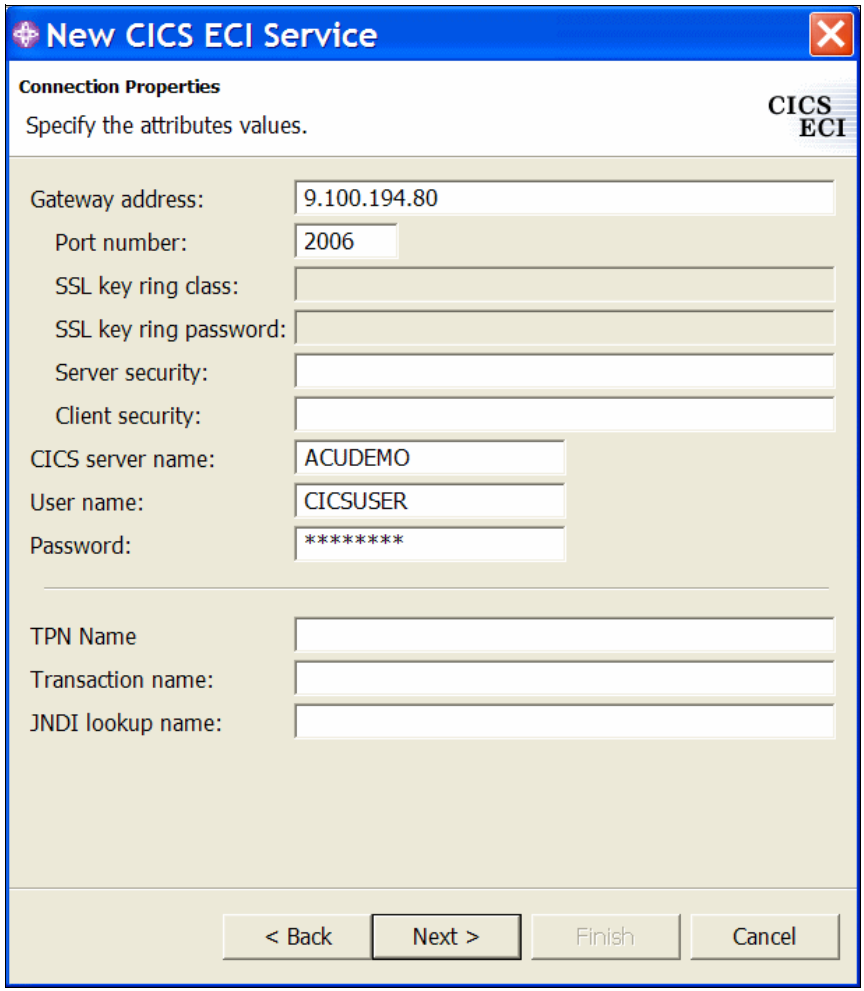


Figure 4-65 Specifying the attributes

Important: The user name and password in the service WSDL file are not encrypted. We remove them from the port definitions as soon as we complete our testing.

In the Gateway address field, we type the TCP/IP address rather than the local host, because for our demonstration environment the back-end TXSeries program to access is deployed under TXSeries for AIX. Therefore, we have to point to the Gateway daemon of the AIX machine rather than using the CICS Transaction Gateway Connector provided with WebSphere Studio Application Developer Integration Edition.

4. Specify the location to store the generated files. In the Service Binding page (Figure 4-66 on page 206), perform these steps:
   a. Make sure that the Source folder name (where the generated service definition files are stored) is correct. In most situations, the source folder is the service project. If necessary, click **Browse** to locate and select this folder.
   b. Click **Browse** to locate and select a package where you want to store the WSDL file. You can also enter a new package name and it is created for you. It is important to specify a package, and not to use the default. Eventually, the package name that you specify is used with the deployment code of the generated EJB.
   c. Specify names for files and service elements. Type a name for the Package and for the Interface file name (in this case, we specify `cics.custdetail` and `CustomerInfo`). This same name is used for the Port type name. The default binding and service names and file names are completed for you. You can modify these default names if necessary.

5. Click **Finish**, as shown in Figure 4-66.

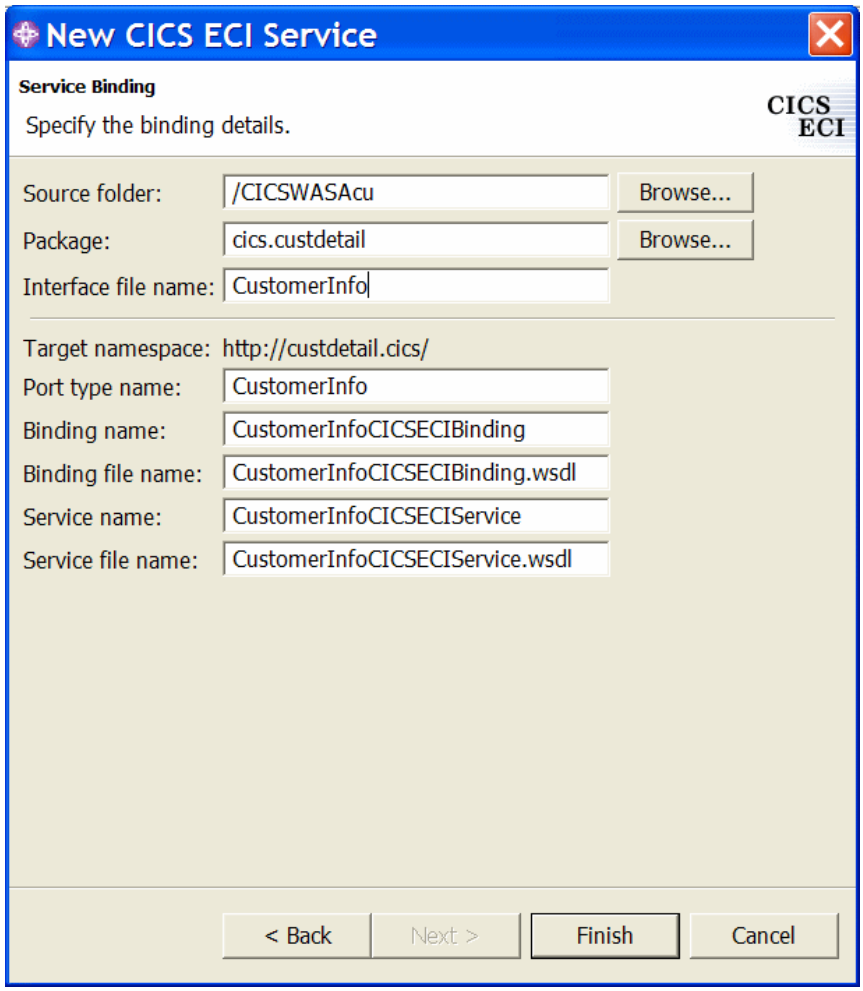


Figure 4-66 Specifying the binding details

At the end of these steps, three WSDL files are generated:

- The Interface WSDL file has a port type with no operations and messages.
- The Binding WSDL file does not have any binding definition for any operation.
- The Service WSDL file contains the required service and port definition.

You have completed only the service and port definitions. You have not yet added any operations, messages, or binding (implementation) information to the service definition.

When the Create Service wizard finishes processing, a dialog displays a message to proceed to create the binding operations in the Binding WSDL file, which is opened in the editor. You should now complete the service definition by using the WSDL editor to add operations and bindings for the CICS ECI service.

Perform the following steps to add the operations, messages, and CICS ECI bindings to the service definition in the WSDL files. The operation and message definitions are usually imported from either the COBOL or the C source program.

After generating the service for the CICS program, the binding file is now open for editing. In the WSDL editor's Graph view, you can add the bindings definition. To complete the service definition, you have to add the required operations and messages for the service, and then create the bindings for them.

Before you specify the bindings for operations, you must add the operations to the interface WSDL file. When importing message definitions from COBOL or C source, the WSDL editor enables you to add new operations from the binding WSDL file, and the new operations are added (and saved) to the interface WSDL file. We demonstrate this in the following steps.

Follow these instructions to complete the service definition:

1. In the WSDL editor's Graph view, select the binding in the Bindings container, right-click, and select **Generate Binding Content**, as shown in Figure 4-67.

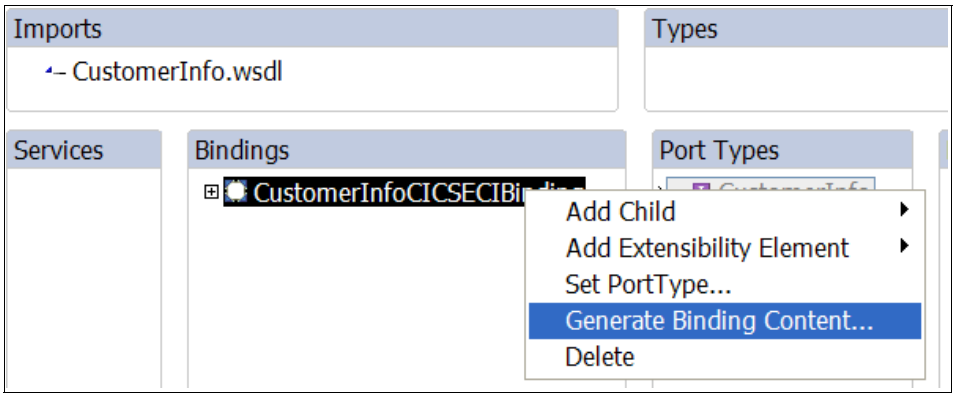


Figure 4-67 Generating the binding content

2. The Binding wizard opens. The Port Type is already specified. CICS ECI is preselected for the Protocol. Under the Add binding operations container, click **Add**.

   The New Binding Operation wizard opens. Next, you define the operation that you want to add.

3. In the first Operation Binding page, complete the following steps:
   a. Enter a name for the new operation (in our example, we type `getCustomerInfo`).
   b. Select the type of operation, which can be either `REQUEST-RESPONSE` or `ONE-WAY`. Typically, you must select `REQUEST-RESPONSE`, which is what we also do.
   c. Click **Next**.
4. On the next page (Figure 4-68), specify the information for the CICS connector. Some default values are provided in the page. You must enter the functionName. This must match with the CICS program name that is invoked for the service (in our case, `CUSTECI`).

   For additional information about these specifications, see the CICS resource adapter's Javadocs. You can also expose `ConnectionSpec` and `InteractionSpec` properties as data in a message. For more information, see *Exploring WebSphere Studio Application Developer Integration Edition 5.0*, SG24-6200.
5. Click **Next** to go to the next page of the wizard.

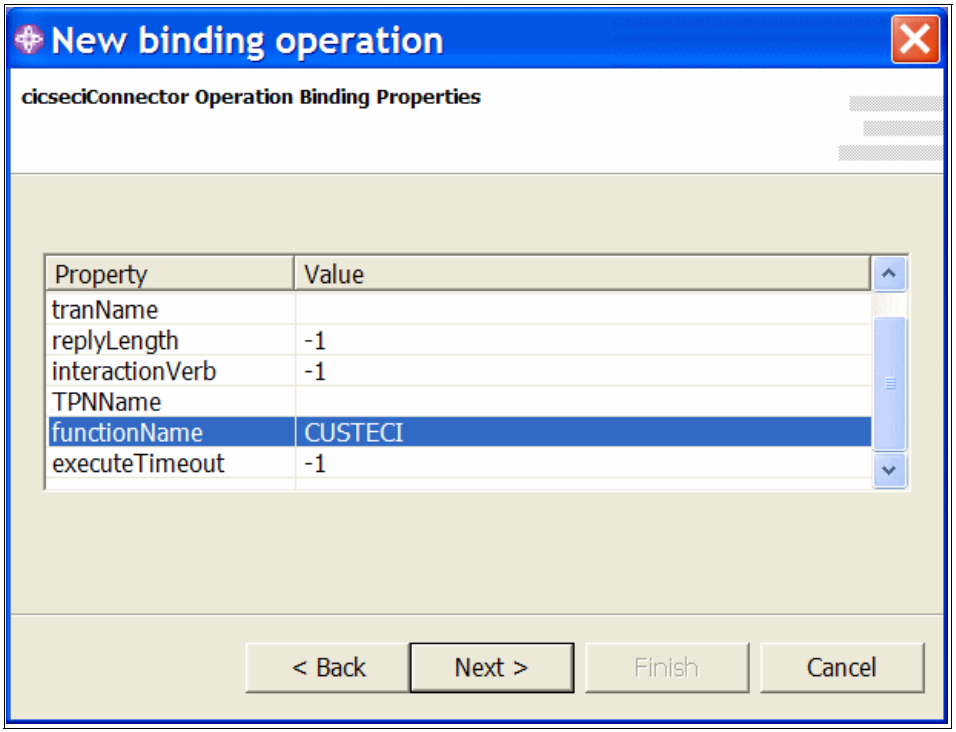


Figure 4-68 New binding operation

6. Specify the input message for the operation. When the input message's definition comes from a COBOL or C program rather than from a WSDL file, you have to click **Import** (rather than **Browse**).
7. A File Selection wizard opens. Select the **Use the input message for output** check box.

> **Note:** For resource adapters, we rarely take messages from a WSDL file using the **Browse** button. However, if this is the case, the physical binding and typemap information must already be available in your binding.

8. Select the COBOL source file (`CUSTECI.ccp`) from the Source files container and click **Next**.

> **Requirement:** If you have not imported the source file into the workspace, you do not see the COBOL or C source file in the Source files container. In this case, you have to follow the instructions in Importing a COBOL or C file, as described in Step 6.

9. In the COBOL Import Properties page, accept all the defaults and click **Next**.

> **Requirement:** When you import definitions from COBOL source, you have to specify information, as shown in Figure 4-69 on page 210.

10. In the COBOL Importer dialog, scroll down the communication data structures and select **DFHCOMMAREA**, as shown in Figure 4-69. Click **Finish**.

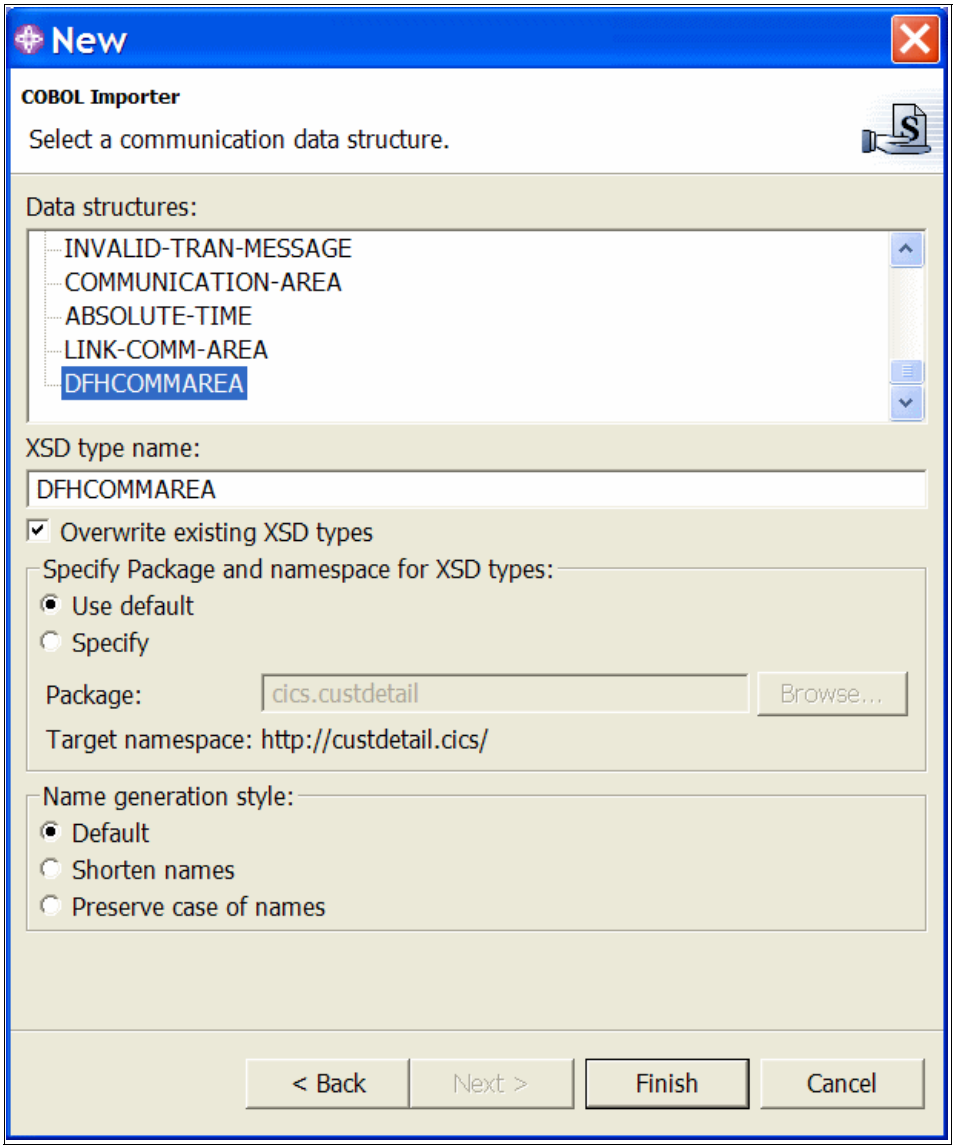


Figure 4-69 COBOL importer

11. Go back to the Operation Binding pane. You see that the Input message has been completed as a result of importing the COBOL. Click the **Next** button in this pane.

12. In the Operation Binding Summary pane, briefly examine the Summary. Note that the customerInfo Operation is being added, in addition to the CustomerInfoRequest message. Click **Finish**.

    You should see the CICS ECI binding definition in the Bindings container, as shown in Figure 4-70.

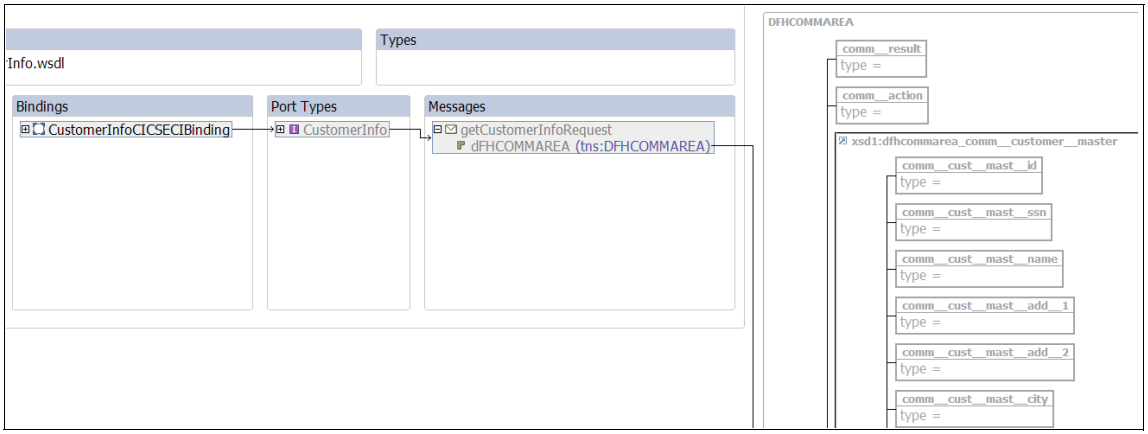


Figure 4-70 CICS ECI binding definition

> **Information:** For CICS ECI resource adapter services, XML Schema Definitions (XSDs) are always inlined in the interface WSDL file.

At the end of these steps, the interface and binding WSDL files are modified for the CICS ECI enterprise service.

> **Remember:** Save (Ctrl+S) the content of the `CustomerInfoCICSECIBinding.wsdl` file before generating any deployment code.

Generating the deployment code for the enterprise service

Now, you have to generate the deployment code. The Generate Deploy Code wizard generates a stateless session bean to pass the client request to the service and handle the returned response from it. The wizard also generates deployed classes that enable your session bean to operate on an EJB server, such as the Application Server. The Generate Deploy Code wizard generates an EAR file for the service, and this EAR file is what we deploy to the run time.

To generate the deploy code for the CICS ECI service, follow these steps:

1. Expand the **CICSWASTX service** project and **cics.custdetail** package. Select the service binding file named **CustomerInfoCICSECIService.wsdl**.
2. From the tool bar, click the **Generate deploy code for a service** icon.
3. The Generate Deploy Code wizard opens. In the Deployment window, the WSDL file, service name, and port name have default values based on the service binding file that you selected previously. Because you are creating a service that uses EJB, you have to select **EJB** from the Inbound Binding type drop-down list. Default values for Enterprise application project and EJB project are already specified.
4. Select the **Generate helper classes** check box and click **Next**.
5. The Inbound Service Files window enables you to specify the names, file names, and location of the EJB service binding and service files. Accept the generated defaults, because they are based on the interface file name and your selected inbound binding type. Click **Next**.
6. In the EJB Port window, leave the default for the JNDI name. This JNDI name is for the EJB binding. Again, accept the default. Click **Finish**.

> **Tip:** To get more help, press the Tab key to get to the entry field or the button in the wizard's page and press the F1 key.

The CICSWASTXEJB project contains all of the resources for EJB applications, including the following components:

- Stateless session bean (`CustomerInfoService`)
- Deployment descriptor
- Remote interface
- EJB home
- Service definition file (`CustomerInfoEJBService.wsdl`) that describes the generated session bean
- Service project (`CICSWASTX`) that contains a service definition

  This project is compressed as a JAR file and placed in the EAR project (`CICSWASTXEAR`), together with the `CICSWASTXEJB` project's JAR file.

At this point, you have an EJB session bean that is able to access the CICS Server and use the **`DFHCOMMAREA`** as the input/output (I/O) parameter. To test this EJB using the JCA connectors, you have to create and configure the WebSphere Application Server inside the WebSphere Studio Application Developer Integration Edition test environment.

Testing the code using the WebSphere Studio Application Developer Integration Edition test environment

You can create a local test environment server and a server configuration in the WebSphere test environment of the WebSphere Studio. You can then deploy your business application to that server for local testing. You can also create a WebSphere Remote Server and server configuration, which you can use to deploy your business application, and then test it remotely in one of several ways.

This section explains how to use the test environment to test the TXSeries enterprise services application that you have just created.

Adding a server and server configuration

This step adds a server and a server configuration to run the generated session bean. A server identifies the runtime environment that you want to use for testing. A server configuration contains information that is required to set up and publish a server. (Skip this step if you want to use an existing server and server configuration.)

The server can be created automatically when testing components, but you can create a server and configuration manually:

1. In the Services view, perform the following steps:
   a. Right-click the server project and select **New** → **Other**.
   b. In the left pane, select **Server**.
   c. In the right pane, select **Server and Server Configuration**.
   d. Click **Next**.
2. The Create a New Server and Server Configuration wizard opens. This wizard creates a new server and server configuration at the same time. Type `WASAS` as the server name, and follow the instructions in the wizard to specify the details of the server and server configuration that you want to create.

3. In the Create a New Server and Server Configuration window, in the Server type field, select **Integration Test Environment**, as shown in Figure 4-71. Click **Finish**.

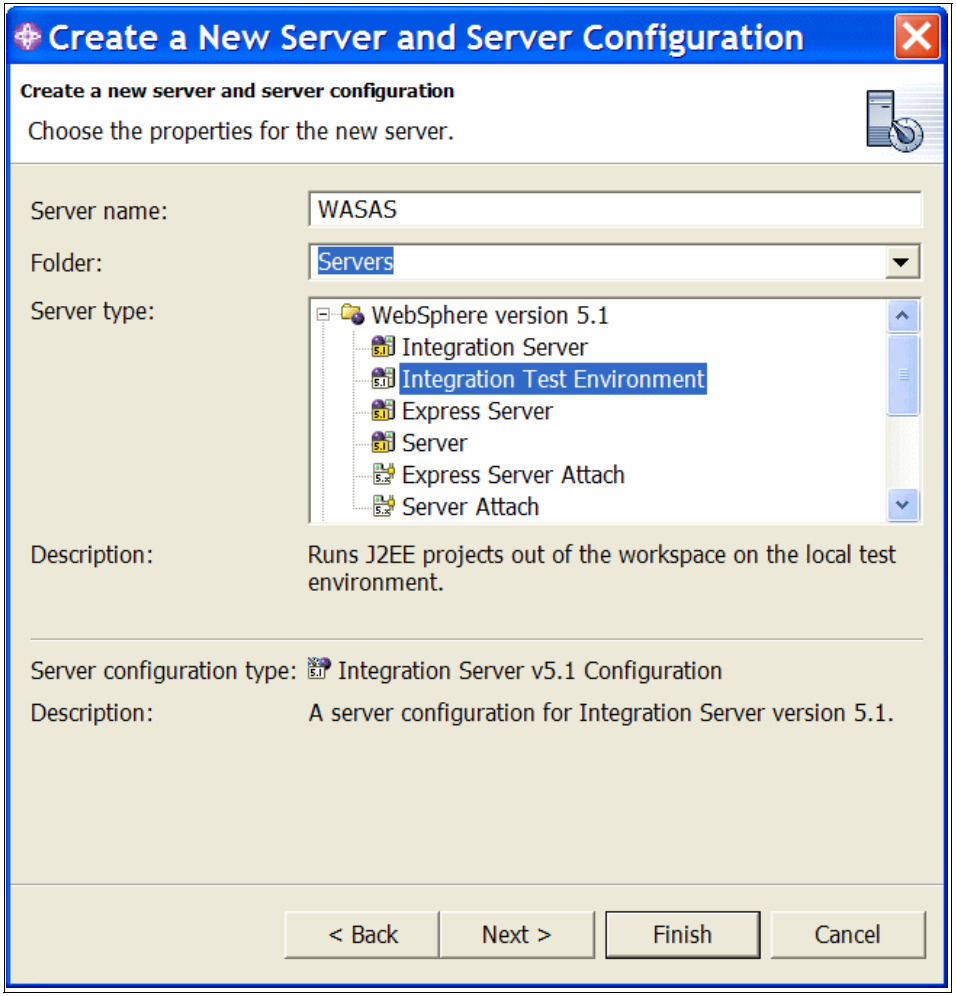


Figure 4-71 Choosing the properties for a new server

Open the Server perspective and you can see the following information:

- The new server and configuration folder under the Servers project folder in the Navigator view
- The server in the Servers view and the Server Configuration view
- The server configuration in the Server Configuration view

Adding Java Authentication and Authorization Service authentication to the server configuration

WebSphere Application Server V5 and later fully support the Java Authentication and Authorization Service (JAAS) architecture. It extends the access control architecture to support role-based authorization for Java EE resources, including servlet, JSP, and EJB components.

> **Remember:** This activity depends on your specific environment.

To use JAAS authentication, you have to add JAAS authentication entries to the server configuration:

1. Using the Server perspective and the Server Configuration view, select the server **WASAS**. Double-click to open the server configuration properties.
2. In the server configuration properties, select the **Security** tab. For JAAS Authentication Entries, select **Add**.
3. The Add JAAS Authentication Entry dialog opens. Enter `cicsentry` for the Alias value. Specify the User ID and Password (in our case, `CICSUSER` for both), as shown in Figure 4-72.

   For JCA, the alias name is used as the user ID to obtain a connection to the back-end system. When you add the JCA connection factory, you have to select this alias name for the component-managed and container-managed authentication aliases. For more information about JAAS support, see the Java Authentication and Authorization Service subject using the Help feature.
4. Click **OK**.

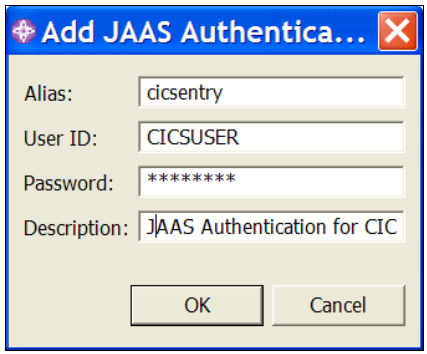


Figure 4-72 Adding JAAS authentication

This step is required because we implement our CICS connection using the JCA adapter. A connection factory for the JCA resource adapter in the server configuration provides the connections to CICS.

When defining the connection factory, specify all of the information required by the resource adapter to connect to the particular instance of CICS. Also specify the JNDI lookup name under which the new connection factory instance is available to components. With this lookup name, the components can quickly make a connection to CICS. At run time, a factory object can generate a connection, locate the CICS server, and start the CICS program at the server.

To add a connection factory, perform the following steps:

1. Using the Server perspective, expand **Servers** in the Server Configuration view. Double-click the server **WASAS** to open it in an editor.
2. Select the **JCA** tab.

   With WebSphere Application Server, depending on what you are administering, you can configure at the cell, node, or server level. Cells contain nodes which contain servers. Values specified at a higher level apply at the lower levels, unless they are overridden at the lower level.

   In the JCA page, you can define a connection factory at the node level, and it applies to all of the servers that it contains. Alternatively, you can define the connection factory at the server level.

   Click **Add**, which is next to the JCA Resource Adapters table.
3. The Create Resource Adapter wizard opens. Select the name of the resource adapter from the Resource Adapter Name drop-down list. (In our example, we select **Resource Adapters Project**.) Click **OK**.
4. In the server configuration editor, select the new **ECIResourceAdapter** in the JCA Resource Adapters table. Click **Add**, which is next to the JCA Connection Factories table.

5. The Create Connection Factory wizard opens (Figure 4-73). Enter the following values:

   a. Name: `CICSECI_CF`

   b. JNDI name: `eis/cicseci`

   c. Container-managed authentication alias and Component-managed authentication alias: `cicsentry` (the same alias that you specified in the Security page).

   Optionally, you can specify other values.

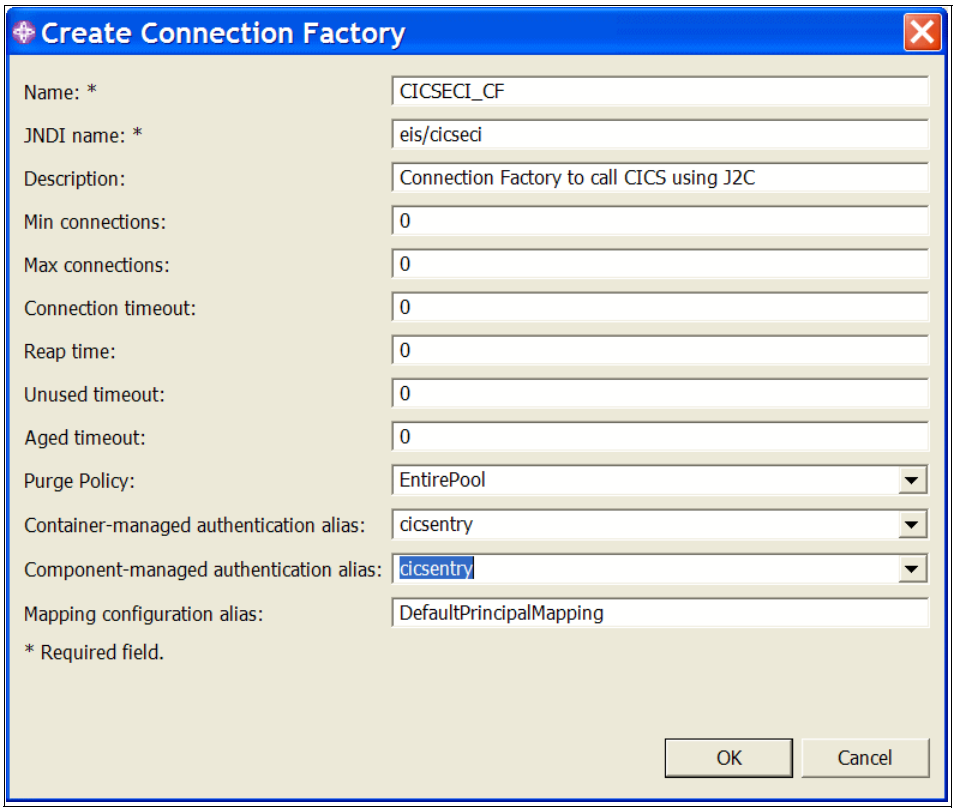


Figure 4-73 Creating the connection factory

6. Click **OK**.

   We later bind this JCA connection factory's JNDI name to the generated deploy code's session bean.

7. In the server configuration editor, enter the connection properties in the Resource Properties table, as shown in Figure 4-74. For this demonstration environment, we enter the following values:
   a. ServerName: `TXDEMO` (the name of the CICS server in the `ctg.ini` file)
   b. ConnectionURL: `tcp://9.100.194.80` (the URL of the CICS Transaction Gateway on the AIX machine to access the server)
   c. PortNumber: `2006` (we leave the default)
8. Close the editor and click **Yes** to save the changes.

Resource Properties:

Name	Type	Value
ServerName	java.lang.String	ACUDEMO
ConnectionURL	java.lang.String	tcpip://9.100.194.80
PortNumber	java.lang.String	2006

Figure 4-74 Resource properties

9. Validate your CICS Transaction Gateway configuration on the AIX machine to ensure that the definitions match.

Binding the connection factory

Now you have to bind the JCA connection factory to the generated (deployment code's) stateless session bean.

To bind the JNDI name, perform the following steps:

1. Using the Business Integration perspective and the Services view, expand **Deployable Services**. Right-click the **CICSWASTXEJB** project, and select **Open with** → **Deployment Descriptor editor**.
2. When the editor opens, complete the following actions:
   a. Select the **Reference** tab.
   b. In the EJB container, expand the EJB project and the session bean until you can see and select **ResourceRef**.
   c. Under WebSphere Bindings, enter `eis/cicseci` as the JNDI name of the JCA connection factory that you have added to the server configuration. See Figure 4-75 on page 219.
   d. Press Ctrl+S to save the changes.

Name: cics/custdetail/CustomerInfoCICSECIService/CustomerInfoCICSEC

Description: CICS ECI Connector Binding.

Type: javax.resource.cci.ConnectionFactory

Authentication: Container

Sharing scope:

WebSphere Bindings

The following are binding properties for the WebSphere Application Server.

JNDI name: eis/cicseci

WebSphere Extensions

The following are extension properties for the WebSphere Application Server.

Figure 4-75 Binding the JNDI name

Testing the session bean

You are now ready to test the EJB session bean that calls the CICS server.

Important:

- Before testing your code, you have to start the TXSeries region and the CICS Transaction Gateway (Gateway daemon and Client daemon):
  - `cicscp start region txdemo StartType=cold`
  - `export PATH=$PATH:/usr/IBM/WebSphere/AppServer/java/bin`
  - `ctgstart`
  - `cicscli -S=TXDEMO`
  - `cicscli -L`
- Remember to also start DB2 in case your program has to access it.

To test the EJB, we use both WebSphere Studio Application Developer Integration Edition and the Universal Test Client tool to make sure that the generated session bean is functional. The Universal Test Client enables us to view and start methods, objects, and classes to test the session bean.

To start and use the Universal Test Client, perform the following steps:

1. In the Services view, under Deployable Services, select the EJB project that contains the session bean. Expand the project until you find the **CustomerInfoService** session bean. Right-click it and select **Run on Server**.
2. In the Server Selection wizard, select the server you created, **WASAS**.
3. Click **Finish**.
4. In the References pane, under EJB References, expand the session bean, and then the home bean. Select the **create** method for the home bean.
5. In the Parameters pane, complete the following actions:
   a. Click **Invoke**. This creates the home object for your session bean and displays a Work with Object button.
   b. Click **Work with Object**. This creates a new bean in the References pane.
6. Expand this new bean to display a list of methods that can be run in the bean, and the method that you want to run. In our case, we select **DFHCOMMAREA getCustomerInfo(DFHCOMMAREA)**.
7. In the Parameters pane, the input field for the selected method is displayed. Expand DFHCOMMAREA and enter the following values, as shown in Figure 4-76 on page 221:
   – For comm__action: 1.
   – For signon__passwd: `CICSUSER`.
   – For signon__user: `CICSUSER`.
   – Click **Invoke** to run the method.

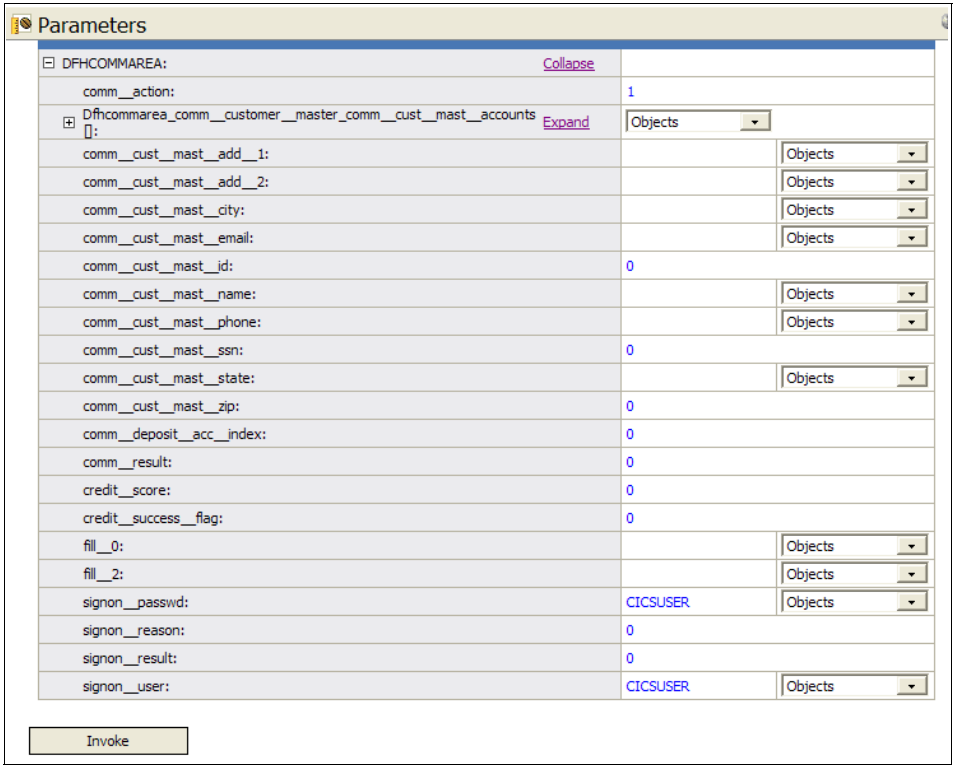


Figure 4-76 Running the method

8. A result is returned to the Results field, as shown in Figure 4-77. Click **Work with Object** again.

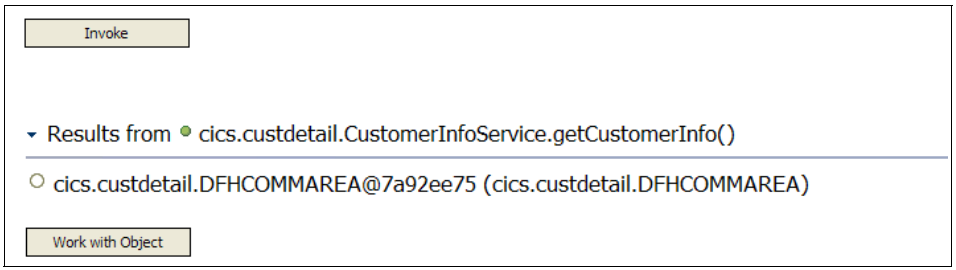


Figure 4-77 Result of running the method

9. In the References pane, expand the **Object References** and also the **DFHCOMMAREA@** entry corresponding to the one returned by the Invoke. Scroll down and select the **int getComm__cust__mast__id()** field.

10. In the Parameters pane, click **Invoke**. You get back the value 111111000, which is the customer ID number for the CICSUSER user. See Figure 4-78.

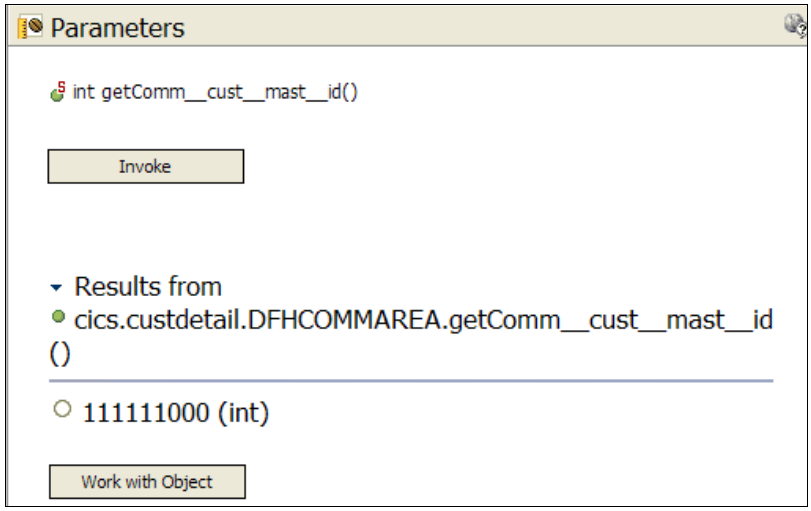


Figure 4-78 Customer ID shown

Completing the enterprise application

To complete the enterprise application so that it is ready for deployment on a WebSphere production system, there are several additional steps to complete:

1. Add variables representing the inputs and outputs to the session bean, expose them as getters and setters, and promote to the EJB remote interface.
2. Create an access bean to simplify client access to the session bean.
3. Complete a Java EE web project that can pass customer details taken from a web page to the access bean, and display the results.
4. Test the whole enterprise application project.

Modifying the session bean

Before creating an access bean and completing the web project, you have to add code to the generated session bean. This code sets the customer number before the CICS call, and retrieves the customer information (name, address, and account details) after the CICS call.

This can be done in many ways. In this example, we choose to provide access to the DFHCOMMAREA structure required by the session bean. Provide this access by defining individual fields for the customer information, and extracting the data from the DFHCOMMAREA into these fields.

To do this, you have to edit the `CustomerInfoServiceBean` Java class:

1. Create variables for each of the customer data items you have to pass to or from the EJB when invoking the `getCustomerInfo` method. See Example 4-6.

   Example 4-6 Variables added to the session bean

```
// Added
private short communicationsResult;
private short communicationsAction;
private int customerId;
private int customerSsn;
private String customerName;
private String customerAddress1;
private String customerAddress2;
private String city;
private String state;
private int zip;
private String phone;
private String email;
private int[]customerAccountID = new int[10];
private BigDecimal[]customerAccountBalance = new BigDecimal[10];
private String[]customerAccountType = new String[10];
private String signon;
private short depositAccountIndex;
private String password;
private short signonResult;
private short signonReason;
private short creditFlag;
private short creditScore;

private DFHCOMMAREA record = new DFHCOMMAREA();
private
Dfhcommarea_comm__customer__master_comm__cust__mast__accounts[]
accounts;
// Added until here
```

2. Add an import statement for `java.math.BigDecimal` to resolve the compile error produced by the introduction of the BigDecimal class.

3. Create the getters and setters for the recently added fields of the Java class by opening the **Outline** view for the class. Select all of the recently added fields (from **communicationResult** to **record**) and select **Source** → **Generate Getter and Setter** from the menu bar.

4. Add the necessary code to initialize the DFHCOMMAREA with the customer details before calling CICS. Retrieve the data after the invocation of the method that calls CICS. This must be done in the `getCustomerInfo` method. The code to be added in the `CustomerDetailsServiceBean` class is shown in bold type in Example 4-7 and Example 4-8 on page 225.

   Make sure that the code is added between the two comments indicating where the code should be inserted. Any code inserted outside of these comments are lost if the EJB is regenerated.

   Example 4-7 shows the pre-execution code.

   Example 4-7 Pre-execution code

```
initialize();

// user code begin {pre_execution}
setRecord(argDFHCOMMAREA);
record.setComm__action(getCommunicationsAction());
record.setComm__cust__mast__id(getCustomerId());
record.setComm__cust__mast__ssn(getCustomerSsn());
record.setComm__cust__mast__name(getCustomerName());
record.setComm__cust__mast__add__1(getCustomerAddress1());
record.setComm__cust__mast__add__2(getCustomerAddress2());
record.setComm__cust__mast__city(getCity());
record.setComm__cust__mast__state(getState());
record.setComm__cust__mast__zip(getZip());
record.setComm__cust__mast__phone(getPhone());
record.setComm__cust__mast__email(getEmail());
record.setSignon__user(getSignon());
record.setSignon__passwd(getPassword());
// user code end

WSIFDefaultMessage aMessage = new WSIFDefaultMessage();
```

Example 4-8 shows the post-execution code.

Example 4-8 Post-execution mode

```
// user code begin {post_execution}
this.record = (cics.custdetail.DFHCOMMAREA)
outputMessage.getObjectPart("dFHCOMMAREA");

setCommunicationsResult(record.getComm__result());
setCustomerId(record.getComm__cust__mast__id());
setCustomerSsn(record.getComm__cust__mast__ssn());
setCustomerName(record.getComm__cust__mast__name());
setCustomerAddress1(record.getComm__cust__mast__add__1());
setCustomerAddress2(record.getComm__cust__mast__add__2());
setCity(record.getComm__cust__mast__city());
setState(record.getComm__cust__mast__state());
setZip(record.getComm__cust__mast__zip());
setPhone(record.getComm__cust__mast__phone());
setEmail(record.getComm__cust__mast__email());

accounts = record.getComm__cust__mast__accounts();
for (int i=0; i<10; i++) {
   customerAccountID[i] =
accounts[i].getComm__cust__mast__account__id();
   customerAccountBalance[i] =
accounts[i].getComm__cust__mast__acc__balance();
   customerAccountType[i] =
accounts[i].getComm__cust__mast__acc__type();
   }
setSignonResult(record.getSignon__result());
setSignonReason(record.getSignon__reason());
setCreditFlag(record.getCredit__success__flag());
setCreditScore(record.getCredit__score());
// user code end
```

5. Promote the generated getter and setter methods to the EJB Remote Interface. To do this, open the **Outline** view. Select the following methods and, using the context editor, select **Enterprise Bean** → **Promote to remote Interface** (this includes all of the `get` methods except for 4):
   - `getCommunicationsResult`
   - `getCustomerId`
   - `getCustomerSsn`
   - `getCustomerName`
   - `getCustomerAddress1`
   - `getCustomerAddress2`
   - `getCity`
   - `getState`
   - `getZip`
   - `getPhone`
   - `getEmail`
   - `getCustomerAccountID`
   - `getCustomerAccountBalance`
   - `getCustomerAccountType`
   - `getSignonResult`
   - `getSignonReason`
   - `getCreditFlag`
   - `getCreditScore`
   - `setCommunicationsAction`
   - `setCustomerId`
   - `setCustomerSsn`
   - `setCustomerName`
   - `setCustomerAddress1`
   - `setCustomerAddress2`
   - `setCity`
   - `setState`
   - `setZip`
   - `setPhone`
   - `setEmail`
   - `setSignon`
   - `setPassword`
6. Save the changes.
7. Deploy the EJB again by using the Java EE view, select the **CustomerInfoService** bean, and selected **Generated Deploy Code** from the context editor.

Creating an access bean

EJB access beans can greatly simplify client access to enterprise beans, and they can also alleviate many of the performance problems associated with remote calls for multiple enterprise bean attributes. Access beans are JavaBeans representations of enterprise beans, and are typically used in client programs that employ JSP files, servlets, or enterprise beans that interface with other enterprise beans.

Access beans shield you from the complexities of managing enterprise bean lifecycles. This means that you can program to enterprise beans as easily as you can program to JavaBeans. This greatly simplifies your enterprise bean client programs, and helps to reduce your overall development time.

To create an access bean for the `CustomerInfoService` session bean:

1. Switch to the Java EE Hierarchy view. Select the session **EJB CustomerInfoService**. Right-click and select **New** → **Access Bean**.
2. The Add an Access Bean wizard opens. Select the **JavaBeans wrapper** radio button. Click **Next**.
3. Make sure that the CICSWASTXEJB project is shown in the EJB Project field. In the Enterprise Bean field, select the **CustomerInfoService** EJB and click **Finish**.

   A class named CustomerInfoServiceAccessBean is generated in the `ejbModule` directory of the session EJB. This class is used in the web project to call the session bean that invokes CICS.

Creating a web project

Now that the EJB has exposed some remote interface methods, and you have created the access bean, the final step is to create a Java EE web project. This project can retrieve customer details from a web page, call the access bean (which in turn calls CICS), and display the returned customer details.

Because web project development is beyond the scope of this book, we created a separate web project, which you can import into the existing enterprise project.

Use the following steps to import the existing web project:

1. From the Studio menu bar, select **File** → **Import**.
2. In the Import window, select **WAR file**. Click **Next**.
3. Using the **Browse** button next to the WAR file field, locate the supplied `\ITSO\SA-Z036-R01\demo\Server\Java\CICSWASTXWeb.war` web archive (WAR) file on the disk. Select the **New** button next to the Project field.

4. In the Dynamic Web Project window, enter a project name of `CICSWASTXWeb`. Make sure that the Project location matches your existing workspace directory. Select the **Configure advanced options** button and select **Next**.
5. On the Java EE Settings page, perform the following actions:
   a. Select the **CICSWASTXEAR EAR** Project.
   b. Set a Web Context of **CICSWASTXWeb**.
   c. Set a Java EE Level of **1.3**.
   d. Click **Finish**.
6. In the Import window, select **Finish** to import the web project.

After you import the web project, you have to update the build path for the web project so that it can reference classes in the EJB module, and locate some Web Services Invocation Framework (WSIF) required classes on the class path.

1. Right-click the **CICSWASTXWeb** project in the Java EE perspective and select **Properties**.
2. In the Properties window, complete the following actions:
   a. Select **Java Build Path**, and the **Projects tab**. Select the **cicseciConnector**, **CICSWASTX**, and **CICSWASTXEJB** projects.
   b. Go to the **Libraries** tab and complete the following steps:
      i. Select the **Add Variable** button.
      ii. Locate the `WAS_EE_V51` directory and click the **Extend** button.
      iii. Locate the `lib/wsif.jar` library and select **OK**.
   c. Repeat Step b for the `lib/wsif-j2c.jar` and `lib/wsatlib.jar` libraries.
   d. Select **OK** to close the properties window.
3. From the menu bar, select **Project** → **Rebuild All** to recompile all of the projects and resolve any compilation errors.

Testing the enterprise project

After you complete the web project, you have to test the whole enterprise application using the WebSphere test environment:

1. In the Java EE Perspective and the Project Navigator view, locate the **Index.html** file. This file might be in `CICSWASTXWeb\WebContent`. From the menu, select **Run on Server**.
2. Studio must redeploy the application, start the test environment server if it is not running.

3. Open a web browser window and open the page shown in Figure 4-79. Enter the following values into the input fields:
   - Signon Userid: `CICSUSER`
   - Signon Password: `CICSUSER`
   - Action: **Read**
   - ID Number: `111111000`
4. Click **Submit**.

> **Tip:** There is no input checking on the web page. Therefore, to avoid error conditions, it is necessary to enter a value in all of the input fields, even though the fields are not required. This is why the fields (labeled SSN to Email) contain values in Figure 4-79.

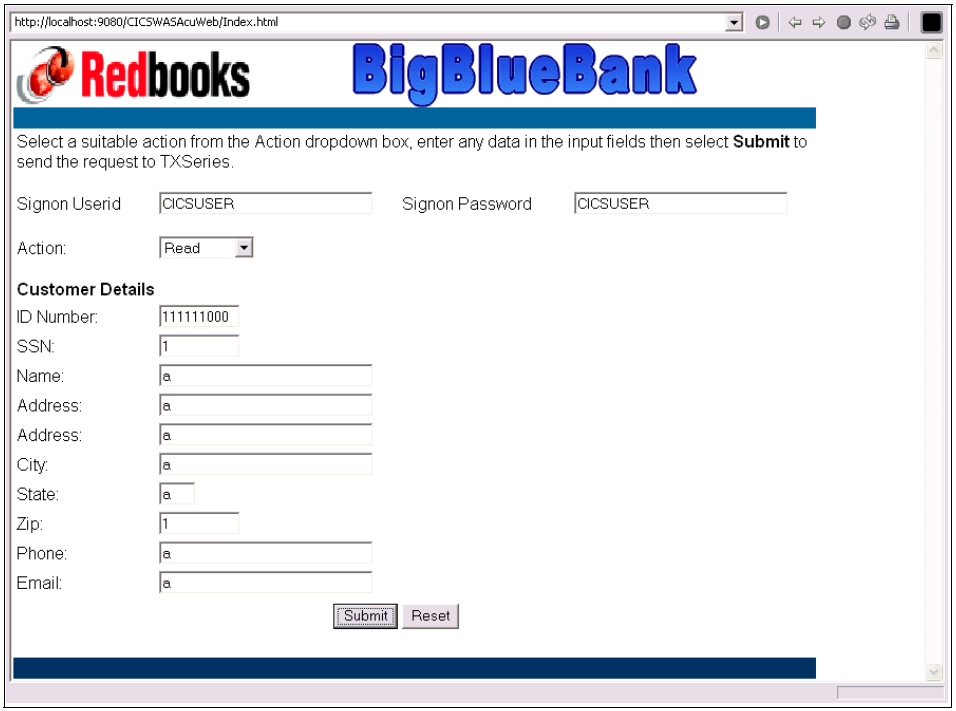


Figure 4-79 BigBlueBank input page

The results page opens, as shown in Figure 4-80.

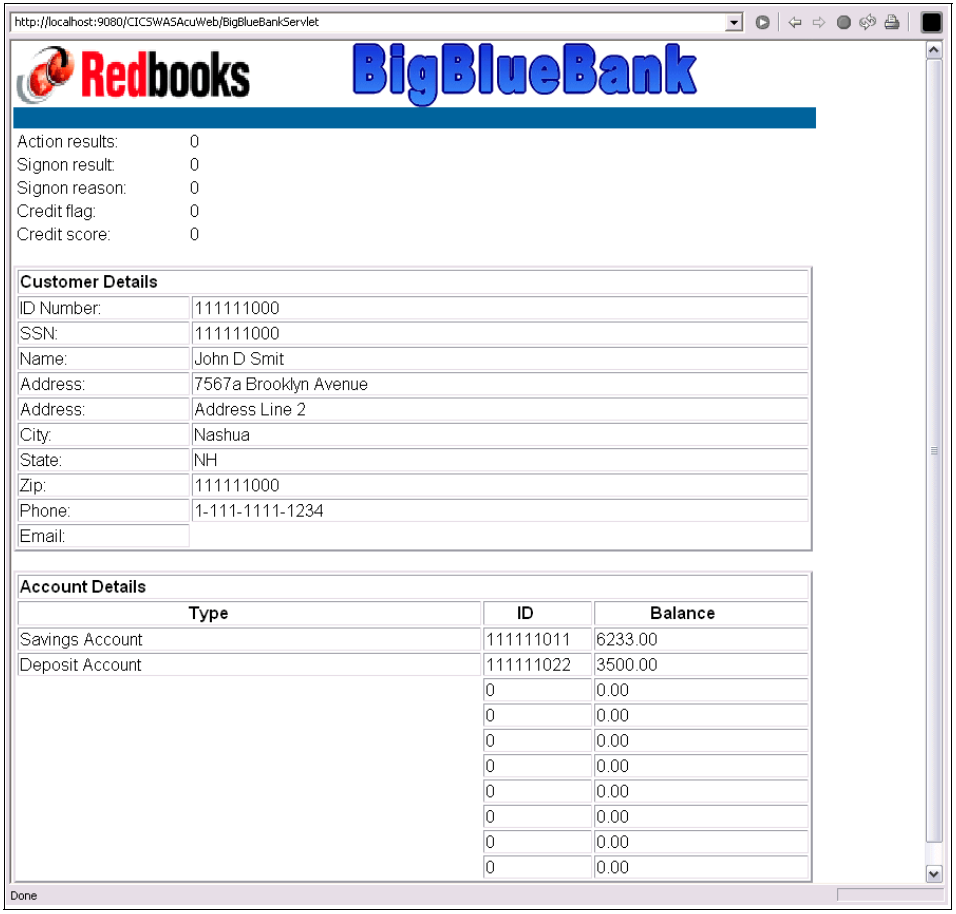


Figure 4-80 BigBlueBank output page

If there are any errors, a message or exception must be written to the WebSphere console.

5. To run the test again, use the **Back** button on the browser, enter some different input values, and resubmit the request.

You now have a working Java EE Enterprise Application that can access a TXSeries CICS application through a web service.

Packaging the TXSeries service for the production server

There are two steps to follow to package the TXSeries service for the production server:

1. Edit the module dependencies.
2. Export the EAR file.

Editing the module dependencies

You must perform this step *only* if you want to test a CICS service on the production server using a SOAP proxy. In this case, we have to edit the web module dependencies.

Information: If your production version of the WebSphere Application Server has the module visibility set to `Application` rather than to the default value of `Module`, you do *not* have to complete this step.

We do not need to run this step for this demonstration. We include it here for reference.

To edit the module dependencies, complete the following steps:

1. Click the **Java EE Hierarchy** tab and expand **Web Modules**. Right-click the web project and select **Properties**.
2. The Properties dialog opens. In it, complete the following actions:
   a. Click **Java JAR Dependencies**, as shown in Figure 4-81.
   b. Select the check boxes for both the JAR file that contains the service interface definition file and the JAR file that is the EJB project.
   c. Click **OK**.

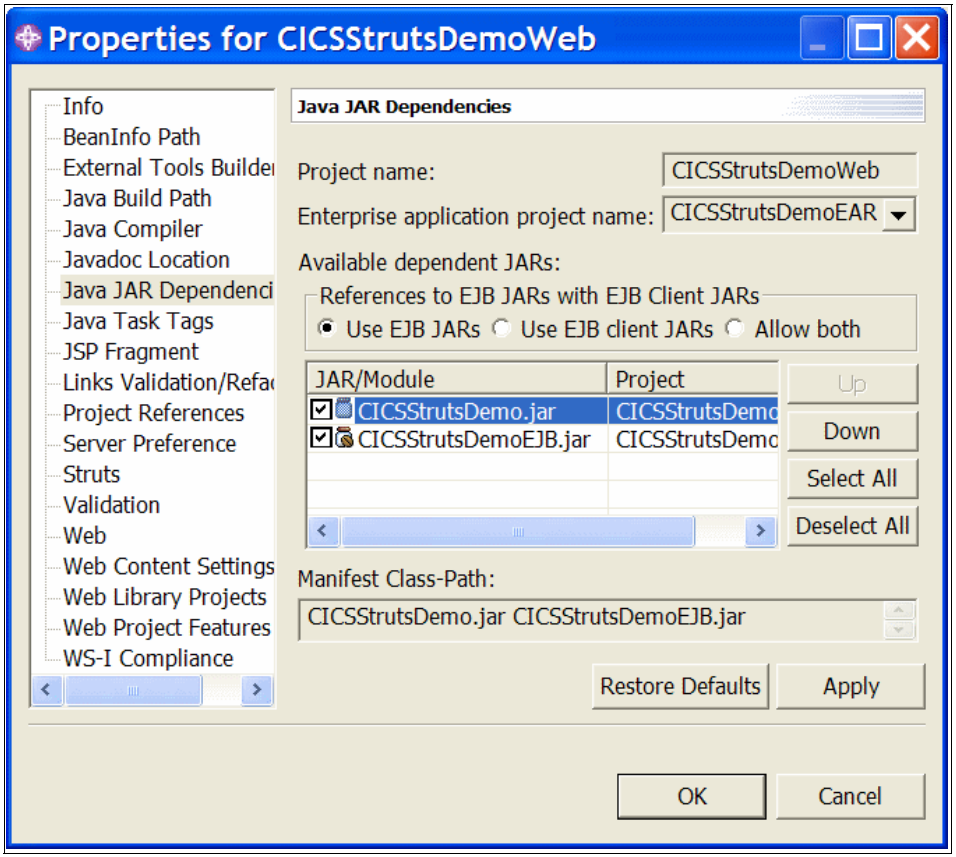


Figure 4-81 Properties for CICSStrutsDemoWeb

Exporting the EAR file

In this section, we export the EAR file that contains our CICS service.

To export the EAR file, complete the following steps:

1. Select **File Export**.
2. The Export wizard opens. In the left frame, select **EAR file** and click **Next**.
3. In the EAR Export window (Figure 4-82), complete the following steps:
   a. In Enterprise Application project name field, select the EAR project (in our case, **CICSWASTXEAR**).
   b. In the Destination field, click **Browse**. Navigate to the directory where you want to export the resources, and then specify the file name that you want for the exported EAR file (in our example, `C:\CICSWASTXEAR.ear`).
   c. Click **Finish**.

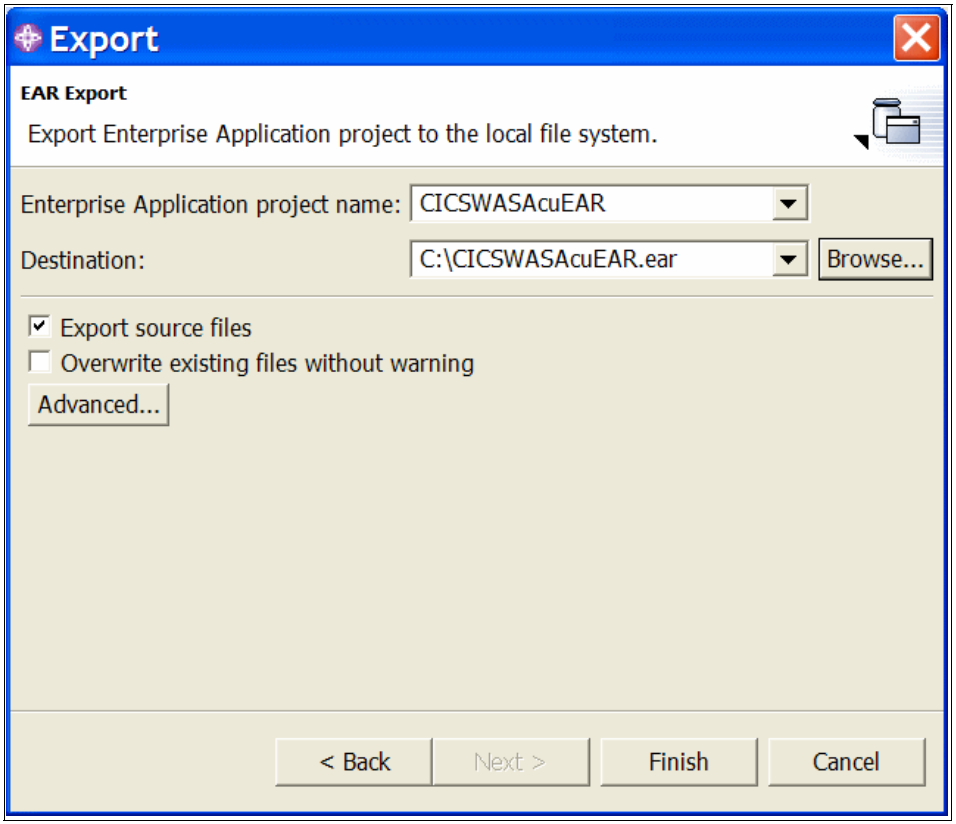


Figure 4-82 Exporting the EAR file

Deploying the CICS service to the production server

To deploy the CICS service to the production server, perform the following steps:

1. Start the administrative console.
2. Install the resource adapter.
3. Install the EAR file.

To complete these steps, see the method described in 4.6.4, “WebSphere Application Server and CICS TG configuration” on page 180. Use the following context root to run the sample application successfully:

`http://<localhost>/CICSWASTXWeb/Index.html`

4.7 TXSeries and Internet Inter-ORB Protocol

Another way to access TXSeries is to use an Internet Inter-Object Request Broker (ORB) Protocol (IIOP) request using the CICS IIOP ORB included in TXSeries. The CICS IIOP ORB enables CICS applications to communicate with applications that use the IIOP protocol, defined as part of the Common Object Request Broker Architecture (CORBA). CORBA clients can communicate with CICS applications, and CICS applications can send requests to CORBA servers.

The Object Management Group (OMG) created the CORBA specification to facilitate the development of object-oriented applications across a network. CORBA objects are standard software objects implemented in any object-oriented programming language.

An ORB mediates the transfer of messages between client programs and objects. When a client program invokes a method on an object, the ORB intercepts the request, and finds an object implementing that method. The result of the method invocation is returned to the client program by the ORB.

From the programmer’s point of view, all of the work appears to be done on one computer system. The IIOP enables communications between different ORB implementations. It is based on TCP/IP, and includes additional message-exchange protocols defined by CORBA.

Information: The CICS IIOP ORB is supported on TXSeries for AIX.

The CICS IIOP ORB is the IBM implementation of the Java ORB, which has been enhanced to run inside CICS. The CICS IIOP ORB supports the following functionality:

- Inbound IIOP from any CORBA-compliant client
- Outbound IIOP to any CORBA-compliant server
- Object interfaces defined in the CORBA Interface Definition Language (IDL)
- Object implementations written in Java
- The use of server groups for load balancing
- The SSL protocol

All of the objects exported from the server are *stateless* objects, which means that no data is held in the object between successive method calls. The object implementation must free all of the resources acquired by the object at the end of every method.

An ORB interacts with application objects, and communicates on their behalf with a collection of system-level resources (CORBAservices Specifications). These can, for example, create components, and name and integrate them into the environment. In general, the CICS IIOP ORB does not support most of the CORBAservices.

This is because some of the functionalities provided by CORBAservices for an ORB are instead provided by CICS for the CICS IIOP ORB, as described in the following list:

- Of the CORBA Lifecycle Services, only the GenericFactory interface is supported. A single generic factory is associated with each CICS IIOP ORB server. When the server is initialized, it writes a reference to this factory into the `/var/cics_regions/region/genfac.ior` file.

  > **Important:** Although the file is written every time the server is initialized, the same string is written out each time. The file usually has to be copied by a client only once, except when a new release of the CICS IIOP ORB is installed, or when the workload management configuration is changed. Except when workload management is used, the generic factory always creates objects in the same CICS IIOP ORB server as the factory.

  This file must be available to any client application program that creates objects on the server:

  – The CORBA Transaction Service is not visible to the server programmer. A new CICS transaction is started with every method invocation. CICS ensures that all of the resources accessed by a method are synchronized when the method returns. Any transaction context passed through from the client is ignored.

- The security support is provided by CICS rather than by the CORBA IIOP Security Service. The IIOP listener can be configured with a user ID. By default, all IIOP requests to a CICS IIOP ORB server run with this user ID.

  In addition, there are two user exits available that can set the user ID under which the request is run. One exit is based on information in the incoming data stream. The other exit enables the user ID to be based on a client certificate, if SSL is being used by the IIOP client. For more information, see documentation specifically for TXSeries.

- The CICS IIOP ORB does not support the following CORBA services:
  - Name service
  - Externalization
  - Identity
  - Persistence
  - Concurrency
  - Interface repository framework
- The CICS IIOP ORB supports workload balancing by dynamically routing IIOP requests. Configuration options enable export and import of interoperable object references (IORs) that contain a server-group host name, port, and object key. For more information, see TXSeries documentation.

TXSeries also includes an IIOP example, the IIOP ORB Sample Bank Application, in the `/usr/lpp/cics/src/examples/orbsamp1` product directory. The banking example program consists of two main parts:

- A traditional CICS application that uses basic mapping support (BMS)
- The `EXEC CICS` API that is written in C

The application consists of the following components:

- The `BNKI` transaction initializes a file with information about several bank accounts. These accounts have numbers in the range of 23 - 30.
- The `BNKQ` transaction queries details in the accounts.
- A CICS program called `CREDCHK` returns the credit rating for an account.
- An implementation of an IDL interface that defines a bank account object. The implementation is written in Java, and runs inside the CICS IIOP ORB. It reuses the existing file to hold the persistent state of the bank accounts, and also reuses the `CREDCHK` program to obtain credit ratings.
- A Java client application that uses the Java runtime environment (JRE) client ORB. This application creates bank account objects and displays information about them.
- A CICS BMS transaction that provides a user interface to the DFHORBUP program, used when a Java implementation class is updated.

Files provided

The banking example application is in the following four directories:

- The `$CICS_install_dir/examples/IIOP/idl` directory contains the IDL definition of the classes.
- The `$CICS_install_dir/examples/IIOP/server` directory contains everything required to build and install the banking server-side applications.
- The `$CICS_install_dir/examples/IIOP/javaclient` directory contains everything required for the Java client application.
- The `$CICS_install_dir/examples/IIOP/classui` directory contains what is required for the UI to the `DFHORBUP` program.

We advise you to use the provided makefiles to compile and run the application because, to build the different parts, these makefiles create several subdirectories under the `/var/cics_regions/region/classes` directory to hold Java classes. In addition, they create files in these subdirectories, and also in the `/var/cics_regions/region/bin` and `/var/cics_regions/region/maps/prime` directories.

Important: You must ensure that these directories and the files in them are accessible by the CICS user. The accessibility of these directories depends on the user under which the makefiles are run, and on the value of **`umask`** at that time.

Installing and configuring CICS IIOP ORB

You must perform all of the steps listed in this section on the server system. The CICS IIOP ORB is supported with the following prerequisites:

- Java software development kit (SDK) installation (Java supplied or supported with the TXSeries version that you are using)
- `JAVA_HOME` setting to your Java SDK installation
- `JAVA_HOME` setting inside the environment file of your region (the `/var/cics_regions/<region_name>/environment` file)

To install a CICS IIOP ORB server, you have to complete the following tasks:

1. Install the IBM Java SDK.
2. Install the IBM Java ORB.
3. Install TXSeries according to the version-specific documentation.
4. Create a TXSeries region in the normal manner.

To configure a CICS IIOP ORB server, perform the following steps:

1. You have to add one listener definition. Use the following command:

   ```
   cicsadd -c ld -r <region-name> CICSIIOP Protocol=IIOP
   TCPService="servicename" CICSuserid="userid"
   ```

   > **Requirement:** If SSL support for IIOP is required, you have to define two IIOP listeners, one for the standard IIOP TCP port and the other for the IIOP SSL port.

   For information about resource definitions to add for the CICS IIOP ORB, see the Administering task for your version of TXSeries:

   http://www.ibm.com/support/knowledgecenter/SSAL2T/welcome

2. When configuring the CICS IIOP ORB server, consider the following details:
   - According to the Bank application requirements, three Object Definitions (OD) must be created in the CICS region. The makefiles perform this task.
   - In the Region Definition (RD) field, the existing **`ClassTableSize`** RD attribute has an extra value at the end for the new OD resource definition. The default is `20`.
3. As already mentioned, by default, all IIOP requests run under the CICS default user, CICSUSER. You can change the user ID under which they run by specifying a different user ID on the **`cicsadd`** or **`cicsupdate`** command used to configure the CICS IIOP LD. If you do this, you must ensure that this user ID is defined to CICS. It does not require a password.
4. By default, IIOP (which includes IBM JDK) uses ORB-controlled threads to start the objects. However, for CICS, the objects have to be started from the CICS application server process main thread. To make it start the objects from the main thread, you must set the following attribute in the `region.properties` file located under the `/var/cics_regions/<region_name>` directory:

   ```
   com.ibm.CORBA.ORBPluginClass.com.ibm.cics.iiop.transport.ThreadPoolImpl=""
   ```

5. After you complete the previous steps, perform a cold start of the region using the following command:

   ```
   cicscp -v start region <region_name> StarType=cold
   ```

   When the CICS region comes up, it creates a cicscb listener. When the cicscb listener is initialized, it generates a `genfac.ior` file in the `/var/cics_regions/<region_name>` directory.

> **Tip:** Copy the `genfac.ior` file to the machine or to the directory where the Java Client is compiled and run.

Now you are ready to work with the Sample Bank Application. The makefile in the `$CICS_install_dir/examples/IIOP/server` directory builds everything required for the CICS server side:

1. Before issuing the **make** command, you must set up your environment by running the AIX setenv.aix script:

   ```
   cd /usr/lpp/cics/src/examples/orbsamp1/server
   ksh setenv.aix
   ```

2. Edit the makefile to define the three macros at the beginning of the file. Specify the following information:

   – The CICS region name
   – The Structured File Server (SFS) name
   – The name of the SFS volume where the file is

   The SFS name takes the form `/.:/cics/sfs/<host name>`. You can list the SFS volumes using the **sfsadmin list lvols** command. You also have to be sure that the SFS is started, because SFS commands are used to create the file to hold the bank account data.

3. After the three macros, set the `JAVA_HOME` variable to your Java installation, and modify the makefile accordingly. Otherwise, you get an error when the makefile tries to find both the **idlj** and the **javac** commands.

4. Enter the **make** command to build all the programs. You can also use the **make** command to install and uninstall all of the CICS resources required for the application on the server.

5. Type the **make install** command to install all the resources in the default directories.

6. Perform a cold start of the region.

7. To run the existing application, use the **BNKI** transaction to initialize the accounts, and **BNKQ** to query them. The accounts created are numbered in the range 23 - 30.

You must perform all of the steps listed in the next section on the Client site, or in the Client directory if you are trying the sample on a single system. For the client application, consider the following information:

- The source of the Java client application is provided in the `$CICS_install_dir/examples/IIOP/javaclient` directory. It is called `BankClient.java`.
- The IDL definition of the server classes in the `$CICS_install_dir/examples/IIOP/idl` directory is also required.

Perform the following steps when building and running the Java client application:

1. Run the setenv.aix script using the following commands:

   ```
   cd /usr/lpp/cics/src/examples/orbsamp1/javaclient ksh setenv.aix
   ```

2. Compile the client application using the makefile. Also ensure that the `/usr/lpp/cics/classes` directory is included in your **`CLASSPATH`** environment variable:

   ```
   export CLASSPATH=$CLASSPATH:/usr/lpp/cics/classes
   ```

3. Before running the client application, obtain the `genfac.ior` file containing the object reference to the generic factory. Copy it from the `/var/cics_regions/<region_name>` directory into the current directory.
4. Verify that the account file is populated with data. Do this by running the **`BNKI`** transaction installed as part of the traditional application.
5. Run the `BankClient.java` program:

   ```
   chmod ugo+x BankClient.java
   BankClient.java
   ```

6. After you run the sample application, you can choose to uninstall all of the resources by entering the **`make uninstall`** command.

4.8 TXSeries and non-CICS applications

Application programs in TXSeries can be easily accessed by a client application that is not itself running inside TXSeries region. For example, a normal UNIX C program can call a program in TXSeries using CICS Universal Client APIs. The CICS Universal Client enables users to access transactions and programs on the entire family of CICS application servers from their desktop. It can communicate with multiple CICS servers using various protocols.

The protocol choice depends on the type of CICS that you have to access. For TXSeries, the protocol to use is TCP/IP. Use the configuration tool to determine the settings for client operation, and to identify the associated servers and protocols used for communication.

External access interfaces enable applications not based on CICS to access and update CICS resources by calling CICS programs, or by initiating CICS transactions. In terms of these interfaces, it is important to understand that CICS Universal Client supports three different types, described in the following section:

- External call interface
- External presentation interface
- External security interface

External call interface

The ECI enables a user application to call a CICS program synchronously or asynchronously. It enables the design of new applications to be optimized for client/server operation, with the business logic on the server and the presentation logic on the client. Using the ECI, you can reach the business logic part of the TXSeries application and exchange a COMMAREA.

External presentation interface

The EPI enables a user application to act as a logical 3270 terminal, and therefore to control a CICS 3270 application. It enables modern technologies, such as graphical or multimedia interfaces, to be used with traditional CICS 3270 applications. Using the EPI, you reach the presentation part of the TXSeries application and exchange a 3270 data stream.

External security interface

The external security interface (ESI) enables user applications to verify and change passwords for specified user IDs that are managed by an external security manager (ESM) on a TXSeries server.

The CICS Universal Client supports the following languages:

- COBOL
- C Language
- C++
- Component Object Model (COM)

Restriction: COM classes are provided for the CICS ECI, EPI, and ESI functions on Windows. However, they are supported *only* for use with Visual Basic and VBScript.

4.9 TXSeries and Host Access Transformation Services

Another type of client than we can use to access TXSeries is IBM WebSphere Host Access Transformation Services (HATS). HATS provides the required tools quickly and easily extend host applications to customers and employees. HATS transforms traditional text-based interfaces, such as 3270 and 5250 green-screen applications, into web, portlet, rich client or mobile device UIs.

It also extends 3270, 5250 and virtual terminal (VT) applications as standard web services. With Rational HATS, you can easily convert traditional text-based host application panels (CICS and non-CICS) to a web *look and feel* (GUI). HATS can also make it easy to improve the work flow and navigation of these traditional applications, without any access or modification to the source code.

The HATS rules-based transformation engine makes it possible to extend traditional applications to the web within hours of installing the software. It is a zero-footprint, zero-download, web-to-host solution. The only software required on the client is a web browser.

HATS provides a strategic platform for enhancing traditional applications, using open, industry-standard technologies, essentially bridging the gap between new and old programming models. The product enables the rapid deployment of traditional applications through a browser, therefore eliminating both the requirement for terminal emulators, and for modification of the traditional code.

Making use of the value of WebSphere Studio and WebSphere Application Server, HATS enables the creation of server-side Java EE applications without requiring any Java programming skills. The value is extended to developers, who can use the many features found in these products.

In addition, by providing a web-based interface to traditional applications, it is also possible to increase productivity and reduce training costs. When used in an SOA, HATS can facilitate improved profitability and speed to market. With HATS, you can realize appreciable business benefits with minimum investment and risk.

A HATS solution fundamentally consists of three components:

- The HATS developer, who is responsible for creating the HATS application
- The HATS server, which is the WebSphere Application Server that runs the HATS application
- The client workstation, which uses a browser to access the HATS application

The power of HATS lies in its ability to accurately recognize the components of the host panels, and transform them in real time to a web interface according to a set of predefined rules. It is easy to customize the rules based on the specific application requirements. HATS can add various elements to the host panels, such as drop-down lists, hot links, tables, buttons, valid-value lists, tabbed folders, and graphs.

It can also add HTML elements, such as logos, graphics, backgrounds, and web links. HATS has macro support that provides programmed navigation through multiple host panels to improve productivity, and to make it easy to use traditional applications.

Figure 4-83 shows an example of HATS architecture.

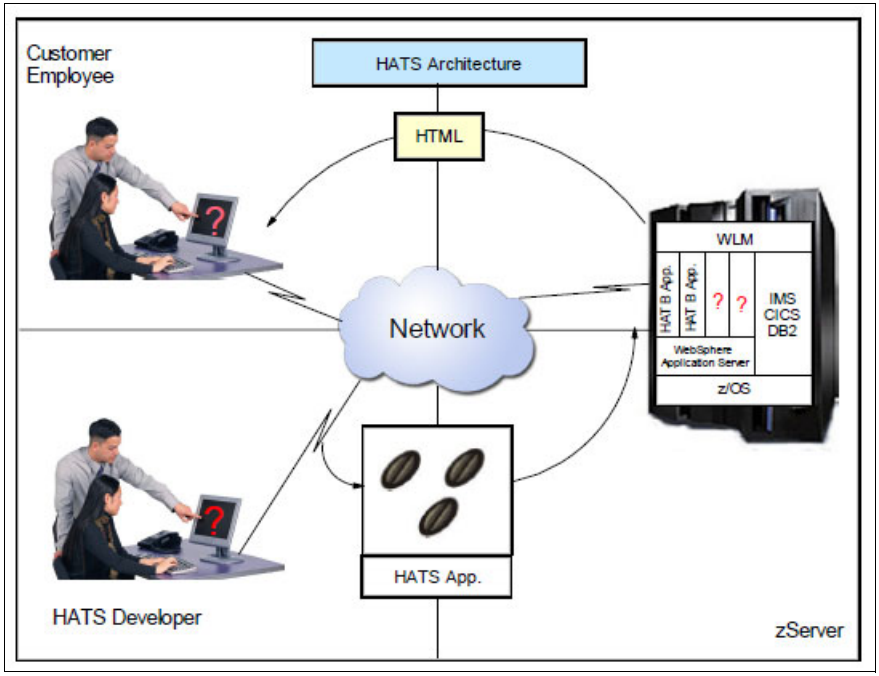


Figure 4-83 HATS architecture example

In Figure 4-84 on page 244, HATS is used to access CICS on zSeries, with WebSphere Application Server and CICS installed on the same mainframe system. However, if we substitute CICS host with TXSeries and remove WebSphere Application Server, installing it on a distributed machine, the architecture is still valid, and HATS works in the same way.

Figure 4-84 is an example of screen scraping using HATS. It shows how a traditional *green screen* appears after using HATS. The result is obtained without modifying a single line of code.

Figure 4-84 Screen scraping

For further details about IBM WebSphere HATS and its features, see the product-specific documentation on the following website:

http://www.ibm.com/support/knowledgecenter/SSXKAY_7.5.0/com.ibm.hats.doc/doc/getstarti03.htm%23gsintro?lang=en

4.10 Security

In this section, we provide a brief outline of the components within TXSeries that are used to establish a secure environment, and the options available. We also briefly describe how to use Lightweight Directory Access Protocol (LDAP) and IBM z/OS Resource Access Control Facility (RACF).

For details about what actions we performed and the parameters we set for the BigBlueBank demonstration system, see Chapter 8, “The BigBlueBank sample application” on page 381.

4.10.1 External authentication manager

TXSeries has a component called the external authentication manager (EAM) that performs your security checking. The external authentication products can be either RACF or any other security product that you are using, through LDAP. The EAM is a predefined interface module that is parameter-driven. You have to customize the parameters accordingly.

There is a sample program supplied by TXSeries for using RACF through an LDAP server. If you want to use a DB2 data store through the LDAP server, you need to customize this program accordingly. The following prerequisites are required for using the sample program:

- z/OS running RACF V1.7 or later
- LDAP server running on z/OS and configured to run in Security Database Manager (SDBM) mode
- IBM Directory Server V5.1 client installed on the machine running the TXSeries region, with EAM configured on TXSeries

For further information about the EAM and its configuration, see the supplied readme files, and product documentation and manuals.

4.10.2 Lightweight Directory Access Protocol configuration

As mentioned previously, the LDAP server has to be configured to use SDBM mode to enable the RACF interface. The LDAP configuration should also include a listener so that LDAP clients can communicate with the server through a given port address. The default address ID is 389, which has no encryption.

It is possible to use SSL through LDAP, but this requires configuration at both the server and client ends. For more information about LDAP, consult the LDAP product documentation and manuals.

4.10.3 Resource Access Control Facility integration

RACF is a security product supplied by IBM, and is a z/OS subsystem. It is used to control user access to resources. TXSeries can use the RACF database to secure its resources using EAM and the LDAP server. To facilitate this, two new RACF attributes have been added to RACF version 1 release 7, which are used by TXSeries:

- TSLKEY
- RSLKEY

TSLKEY and RSLKEY correspond to the LDAP-style attributes of racfTslKey and racfRslKey. These are the transaction-security-level-key and resource-security-level-key.

The TSLKEY is assigned to each transaction, and for users to access that transaction, they have to be assigned the corresponding key. The RSLKEY works on the same principle as the TSLKEY, except that the key is assigned to a resource, so that a user assigned with the same key can access that resource. For further information, see the RACF product documentation and manuals.

4.10.4 The structure of TXSeries and the external authentication manager component

When the sample program is used, the overall structure can be seen in Figure 4-85. This shows how TXSeries CICS running the EAM, LDAP client, LDAP server, and RACF are structured.

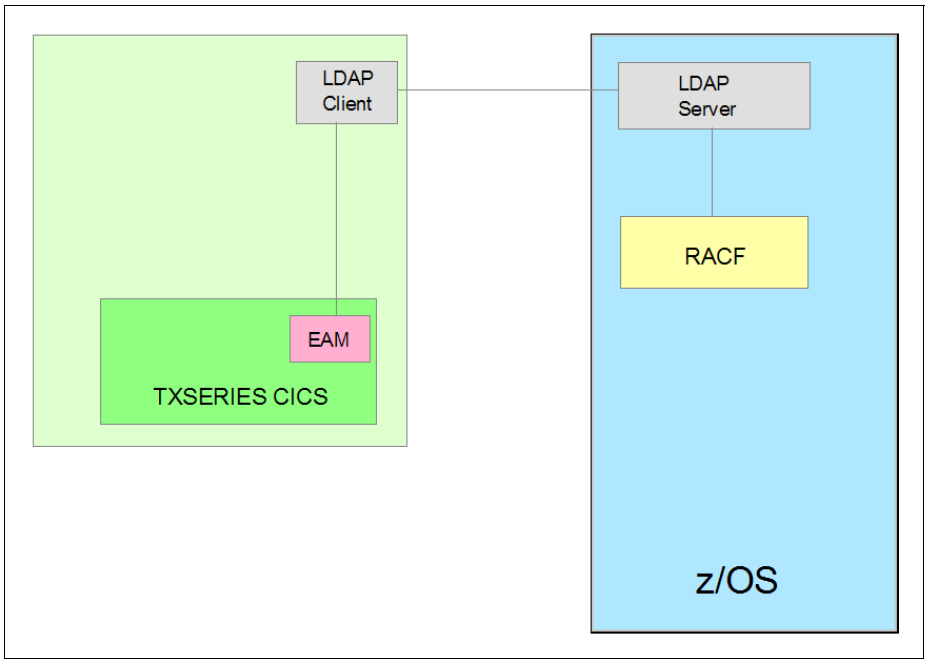


Figure 4-85 EAM, LDAP, and RACF

5

System customization

When implementing IBM TXSeries on a real application system, it is important to customize TXSeries to fit the application model. This chapter describes system customization-related issues related to capacity planning, backup, recovery, high availability (HA), user exits, monitoring and statistics, and transaction tracking.

This chapter provides information about the following topics:

5.1 Introduction to capacity planning

Capacity planning is the process of determining the hardware and software resources that are required to adequately meet an application's service-level requirements. These resources include central processing unit (CPU) power, memory, storage space, and network bandwidth. As a capacity planner, you have to evaluate the current trends in resource usage, and plan for growth and changes before the application faces resource problems.

This section introduces the capacity planning method for TXSeries. Capacity planning must be an ongoing process, with the long-term objective of providing services to the user in a cost-effective manner. Capacity planning managers are challenged to understand their application, usage patterns, and growth patterns to accurately forecast future data processing capacity needs for the business.

Capacity planning must be an integral part of application development. It is a management function closely aligned with the strategic planning for the business. A methodical approach to capacity planning can yield the following benefits:

- To avoid the inconvenience of running out of resources as the business grows
- To postpone upgrades safely, providing significant financial savings in capital expenditure and revenues
- To avoid overcommitment of resources that might lead to performance problems and service-level failures
- To keep upgrade plans ahead of workload increase, because proactive management optimizes your financial investments
- To evaluate the effect of changes in workloads, and to predict the changes required to system configurations
- To present budgets and purchase justifications by identifying the minimum hardware required to achieve target service levels

Each user's behavior is unique, each application's behavior is unique, and each hardware configuration is unique. The overall performance of an application depends on all three of these factors.

5.1.1 Scalability

Scalability, in the context of this section, means the ability of a TXSeries Customer Information Control System (IBM CICS) application system to accommodate an increasing transaction rate, that is an increasing number of transactions per second (TPS). It also includes the ability to assimilate the additional workload without losing functionality when the application system is changed in size or volume to meet a user requirement.

Typically, the rescaling is to a larger resource (for example, one with more CPU, random access memory (RAM), or disk storage). In the new scaled-up environment, the TXSeries CICS application must not only continue to function well, but it must also take full advantage of the added resources, while not requiring changes to the underlying software.

If the load remains constant, and the transaction rate improves at a constant rate relative to additional resources, the system is said to *scale linearly*. Capacity planning estimates are based on the assumption that the application system scales linearly in terms of resources.

5.1.2 Factors that affect capacity estimates

In most application systems, TXSeries CICS is the hub from which other resources are connected and coordinated. These resources include Structured File Servers (SFS), database managers, other CICS regions, mainframes, and queuing and messaging systems.

Therefore, the ultimate performance of the application system is determined not only by TXSeries CICS, but also by all of the other connected systems. The coordination between TXSeries CICS and the other connected systems must be tuned for optimal performance.

The hardware capacity required to support your application depends on the characteristics of the application and configuration. While planning your capacity requirements, you should consider these factors. Understanding these factors helps you to generate server hardware requirements for your configuration. The following questions elicit some of these factors:

- Is your IBM File Manager well tuned?
- Does your network have enough bandwidth?
- Is your application optimized?
- Are your resource managers a limiting factor?
- Are your resource managers well-tuned?
- How many transactions are expected to be run simultaneously at peak hours?
- What types of clients are connected to your TXSeries CICS region?
- How many concurrent connections or terminals exist at peak hours?
- Is your CICS region well-tuned?
- Are there any other applications running in your system in addition to your core application?

5.1.3 Methodology

We set up the test machines in a controlled network environment. We optimized and maintained network resources, such as speed, bandwidth, and availability at reasonable service levels, to reduce any network input/output (I/O) bottlenecks.

The following list summarizes the performance tuning that we completed:

- Set appropriate values for **`MinServer`** and **`MaxServer`** in the resource definition.stanza (`RD.stanza`), in accordance with the required TPS
- Optimized SFS with buffer pool size and thread pool size
- Used program caching
- Turned off any superfluous system services and processes
- Used a reasonably quiet network to avoid any network bandwidth bottlenecks
- Turned off TXSeries CICS statistics recording
- Optimized the **`ClassTableSize`** to accommodate sufficient hash for connections and terminals
- Optimized the checkpoint interval
- Used physical separation of SFS logical volumes to reduce I/O bottlenecks

We monitored the resource usages at the server side using operating system (OS) utilities, such as `vmstat`, `iostat`, and `sar` (on IBM AIX). We also monitored the region activities using TXSeries CICS diagnostic tools, and through traces and other logs wherever appropriate.

Preparing the client

In a capacity planning test scenario, it is not practical to configure many terminals and connections from client machines to represent a live application environment accurately. Therefore, we used load simulators to simulate the real users and terminals that perform online transactions. It is possible to use sample programs provided by IBM CICS Transaction Gateway V6.0 (CICS TG), such as `ecia1.c`.

This sample shows an advanced form of using the IBM CICS Universal Client. It queries the Client daemon for a list of available servers, and then a thread per server is created. Each thread runs a user-supplied number of external call interface (ECI) calls on that server asynchronously, and works out the average response time per server. What users have to do is to customize the program ID, user ID, and user password in the source code and compile it.

Figure 5-1 shows a high-level architectural diagram of the test environment. Note that either the IBM DB2 server or the SFS exists for a given test, but not both.

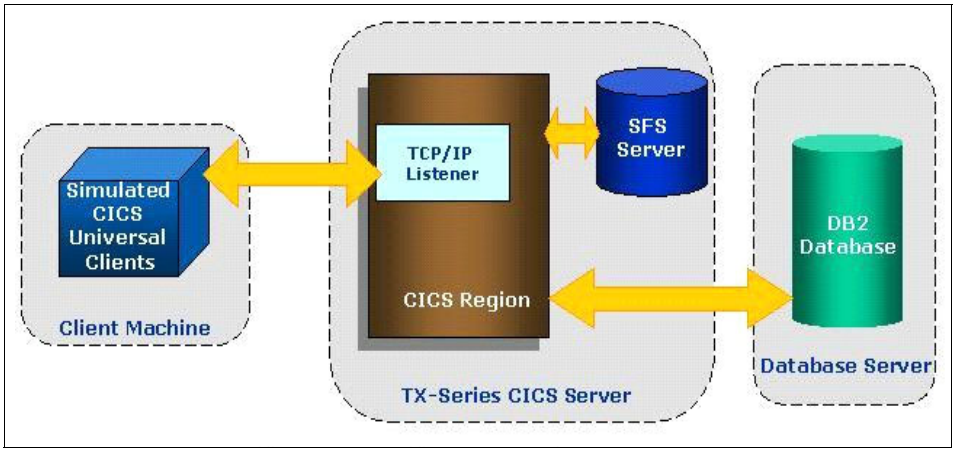


Figure 5-1 High-level architecture of test environment

Preparing the application

The program running on the CICS region is critical to capacity planning. Different kinds of applications result in different capacity planning results. Consider a program with 10 updates to resource managers versus a program with only one query sentence to the resource manager.

A different structure in the resource manager also influences the planning result. Clients must write their own application simulator and resource manager structure to best simulate the future applications. This can be done by monitoring the current system or calculating from the design of future application.

Factors that influence application throughput

The behavior of every user, application, and application deployment environment is unique. If a new system is being established, you can view the results in this test as guidelines for the capacity planning estimates for your environment. If you already have a system running, formulating a new benchmark scenario, which suits the environment, might be more realistic.

There is no substitute for determining a benchmark with the actual production application using production hardware. In particular, your application might reveal subtle contention or other issues that are not captured by these test applications. This also involves analyzing the current usage of the system, and forecasting the system usage over a period of time in future.

This section provides some hints and tips that might help you to perform this job efficiently. We briefly explain a few factors that affect application throughput, and also the parameters that you have to determine from your current system.

Central processing unit

The processing power of the CPU is a key factor that affects your application throughput. Usually, if the application performs many calculations and analysis, it is bound to use more processing power. By monitoring the CPU usage for a reasonable time period, you can determine whether the application under consideration is CPU bound or not.

Typically, if the CPU usage hovers between 70% - 90% on average during the usage monitoring period, you can safely conclude that the application is CPU bound. Performance of the system is optimal if you maintain the CPU usage within 80% of maximum processing power available in the system.

Storage

If the storage available to the application and the CICS region under consideration is not sufficient, it is likely that the system adapts for high paging rates. Storage considerations also include the storage that is available to the SFS logical volumes. TXSeries CICS uses these volumes to store the log information, and for temporary storage of queue information. Storage shortage results in higher response times, and therefore a lower transaction rate.

Disk input/output

Because it takes more time to read data from the disk storage than from memory, disk I/O can adversely affect the performance if the CPU spends time waiting for data to be read from the disk. Ensuring that the disk I/O traffic is optimized minimizes the amount of time spent waiting for the I/O operation to complete.

The monitoring of I/O activities is often done at the OS level, using tools, such as `vmstat`, `iostat`, and `sar`. Typically, you must maintain the wait for I/O within 15% of the total CPU processing time. If it is more than 15%, the application is more I/O-intensive, and you must eradicate this bottleneck.

Physical memory (RAM)

Although the minimum memory required for installing TXSeries CICS is approximately 256 megabytes (MB), this might not be sufficient for a production environment. Allocating a reasonable size of physical memory to areas, such as the region's pool and task private pool, reduces costly virtual memory operations.

Typically, the physical memory usage must be within 70% of total physical memory for optimal performance. Monitoring the memory usage continuously for a reasonable time period can indicate whether the application is suffering from a memory bottleneck.

Network input/output

A typical TXSeries CICS application is distributed in nature. Multiple resources, such as file managers, resource managers, CICS clients, and other CICS regions can be spread over a local area network (LAN) or wide area network (WAN). Therefore, the reliability, speed, and bandwidth of the network are critical to the performance of a TXSeries CICS application.

The data transfer between these distributed resources and the region under consideration depends heavily on the network, which directly affects the performance. If network I/O seems to be the bottleneck, you have to add the necessary resources to the application environment's network subsystem.

Software components

Some of the software components that influence the performance of application throughput are listed in this section. Explaining the details of each factor is beyond the scope of this book. We advise the readers to see the appropriate administration and performance guides to gain insight into each factor:

- Performance of the SFS and other file managers
- Performance of your application
- Performance of resource managers connected to your CICS region
- Performance of your CICS region
- Tuning of your operating system

Software tuning

In this section, we present a guideline for estimating the number of concurrent connections to the region, and the required transaction rate.

Estimating the maximum concurrent connections to the region

Depending on your application and business requirements, you must determine how many connections and terminals are made to the TXSeries CICS region. We suggest that you monitor the connections during peak business hours. The usage pattern during the normal business hours gives an indication about the maximum load that you can expect for the region.

If you make observations for a longer period of time (for example, 5 - 6 weeks), they provide a pointer to the rate of growth in the application system. Based on these values, you can approximately determine the current usage and growth of the system in terms of number of connections and terminals. You have to allocate resources, such as CPU and physical memory (RAM) to the application system based on the number of concurrent connections and terminals.

Estimating the required transaction rate

This refers to the rate at which transactions are run in the CICS region. The transaction rate and response time have an inverse relationship. Therefore, if the application system is intended to perform at a lower response time, the planned transaction rate can be increased.

Typically, an application environment has heterogeneous transaction types, including applications written in C, Common Business Oriented Language (COBOL), C++, and PL/I, with varying complexities. Users can either monitor the system for a period of time to determine the transaction rate, or calculate the ideal value using the application design.

5.2 Backup and recovery

This section describes how to keep your CICS region's configuration and business data from loss or damage through uncontrolled system shutdowns or failures. Typically there are four purposes for the backup and recovery for any kind of server:

- Make synchronous data recoverable. Data, such as the transaction temporary data, and the updates to the resource managers (RMs). TXSeries CICS manages this part of recovery by itself.
- Make asynchronous data recoverable. Data, such as queues and files in SFS.
- Make the server configuration recoverable.
- Failover possibilities. Implement a high-availability environment to make the CICS region restart immediately after a system failover.

5.2.1 Transaction backup and recovery

CICS and the IBM File Manager provide the following mechanisms that you can use in your backup and recovery strategy:

- Logical units of work (LUWs)
- Recoverable resources
- Dynamic transaction backout
- The CICS log
- User journals
- Automatic start
- Transaction logging
- Forward recovery

Synchronization points (SYNCPOINTS)

The end of a LUW is indicated to CICS by a synchronization point, usually abbreviated to *sync point*. A sync point can be caused in the following ways:

- Implicitly at the end of a transaction, signalled by a **RETURN** command at the highest logical level. This means that an LUW cannot span tasks.
- Explicitly by a **SYNCPOINT** command that is issued by the application program at logically appropriate points in the transaction.

An LUW starts when one of the following events occurs:

- At the beginning of a task
- Whenever a transaction issues an explicit **SYNCPOINT** command

An LUW that does not change a recoverable resource has no meaningful effect, either for you or for the CICS recovery mechanisms. CICS does not commit resource changes by a task until CICS reaches a sync point. If CICS interrupts the task processing because of an error of any kind, it backs out the changes made within the LUW experiencing an abnormal end of task (abend).

Two-phase commit

Transaction managers (TMs) and RMs use two-phase commit with presumed rollback, as defined by the referenced open systems interconnection (OSI) distributed transaction processing (DTP) specification (model). In Phase 1, the TM prompts all of the RMs to prepare to commit (or prepare) transaction branches. This queries whether the RMs can ensure their ability to commit the transaction branches. A RM might have to query other entities internal to it.

If an RM can commit its work, it stably records the information it has to remember, and replies affirmatively. A negative reply reports failure for any reason. After making a negative reply and rolling back its work, the RM can discard its knowledge of (the information about) the transaction branch.

In Phase 2, the TM issues all RMs an actual request to commit or roll back the transaction branch, as the case might be. (Before issuing requests to commit, the TM records its decision to commit, and a list of all of the involved RMs.) All RMs commit or roll back changes to shared resources, and then return a status to the TM. The TM can then discard its knowledge of the global transaction.

CICS recoverable resources

To ensure that your business data is updated according to the atomicity, consistency, isolation, and durability (ACID) principles, define the resources in which the data is stored as recoverable resources.

You can define the following resources as recoverable:

- Data files and intrapartition transient data (logically recoverable)
- Auxiliary temporary storage queues
- Autoinstalled terminal definitions
- **START PROTECT REQUESTS** command

For information about the relevant resource definitions, see the TXSeries documentation library on the following website:

http://www.ibm.com/support/knowledgecenter/SSAL2T/welcome

The resource definitions include the following descriptions:

- The `RecoveryStatus` attribute in file definitions (FD)
- The `RecoveryType` attribute in transient data definitions (TDD)
- The `RecoverTerminal` attribute in terminal definitions (TD)
- The `RecoverFlag` attribute in temporary storage definitions (TSD)

During normal operation and shutdown, CICS and the recoverable servers with which it communicates record the information required for restart. If there is a complete or partial error, or a shutdown, this information enables CICS to recover and restart the region. During automatic restart, CICS returns the recoverable resources to their committed states.

If a transaction ends abnormally and the resources affected by the terminating task are recoverable, CICS automatically performs dynamic transaction backout. However, after this, CICS does not automatically restart the transaction. Unless the file is on a DB2 database, where all files are recoverable, you must define a resource as recoverable for CICS to back out the changes.

Automatic start

You can use the `StartType` attribute `auto` setting to start the SFS after a failure. The CICS log is required for a successful restart of the SFS. If the CICS log is not available, a cold start of the SFS is required. Whenever a transaction is started in a CICS region, CICS stores information about it in the CICS log.

The best way to ensure that the log data is recoverable is to use data mirroring for the file system that contains the log data. In practice, this means that the logical volume that hosts the file system must be configured for mirroring. It is best to have a dedicated file system for the log data. This gives you greater control over the allocation of file system storage, and ensures that this storage can be used only for storing the CICS log data.

Remember: It is important to note the following details about CICS logs:

- The CICS log files are not cleared unless a region is shut down normally. Repeated abends can lead to a growth in the size of the log file.
- Do *not* use a Network File System (NFS) for the CICS log.

5.2.2 Asynchronous data backup and recovery

This section describes how to enable your CICS system for recovery if a failure causes the data on the CICS system to be unavailable. Recovering data after an individual transaction fails, or even after some unexpected system shutdowns, can be relatively easy, if you have defined resources as recoverable. CICS and the SFS recover the data in such resources automatically when you restart the system.

All transactions committed before the shutdown are still committed, and all transactions not yet committed are fully rolled back. However, recovering data that is lost from a disk can be more complex, and requires planning and implementing procedures to cope with such a failure. When these types of failures occur, the procedures that enable the recovery and restart of a CICS system depend on the following information:

- The components of that system
- The kind of work performed on it
- The degree of recoverability required

You must consider each individual component of the CICS system, and consider the system as a whole.

Structured File Server recovery

The SFS manages the data associated with CICS files and queues on two logical volumes, one for queue and file data, and one for log data. When you perform a cold start of the SFS, the data on these logical volumes is discarded.

Backing up the SFS logical volumes

Consider the following choices for backing up SFS:

- Online backups using data mirroring. The best way of ensuring that the SFS data is recoverable is to use online data mirroring for the file system containing the logical volumes.
- Choosing an offline backup method for the SFS. There are two options for offline backups of the SFS data:
  - Media Recovery Archiving (MRA), which is provided by Encina
  - Copying the data by using operating system facilities

Backup method considerations

There are advantages and disadvantages associated with the use of MRA or the operating system for backing up the SFS. Consider the following points when you decide which to use:

- MRA enables backups to be taken while the SFS is running. Because the log archive files are continuously written, you do not have to perform frequent backups of the SFS data. The data can always be restored to the level consistent with the latest log archive file.
- Copying the logical volumes has the disadvantage that the SFS must be stopped when you perform the backup, and the data can be restored only to its level when that backup was performed. This means that you have to perform more frequent backups.
- The continuous writing of archive files by MRA can also be considered a disadvantage. It causes some degradation in the performance of the SFS, and the files require disk space. If not properly maintained, the files written by MRA can cause the SFS to end. To avoid problems with the SFS, you have to copy the archive files regularly to ensure that the file system on which they exist does not fill up.
- Copying the logical volumes has the advantage that the data is restored to a known level (its level when the copy was taken). As a result, if the CICS log was saved at the same time, the region can be restarted to a known level.
- With MRA, if the log volume is lost, the level to which the data volume can be restored is not fully defined, although it is probably more recent than the backup of the data volume. This means that the data cannot be restored to a level consistent with a saved CICS log.
- File activity is initially logged by Encina, and is flushed to an MRA log file only when a certain amount of file activity has taken place. Because of this, if you perform an MRA backup at a specified time, there is no guarantee that all of the file activity is included in the backup.
- MRA has the advantage that, if only the data volume is lost, it can be restored from backup to the level it was at when it was lost. In only this respect, it can be considered as an alternative to mirroring the SFS data volume.

Backup using MRA

Consider the following information when backing up the SFS by using MRA. MRA is a function provided by Encina to support the backup and recovery of Encina servers. It enables you to create an offline backup of the SFS with no disruption in service.

The procedure includes the following steps:

1. The server is running with MRA enabled. Create a full backup of the SFS data file logical volume, and either save it to tape or store on another system.

   Restriction: You can create this backup in increments (called a *fuzzy backup*). However, restoration is not possible until a full backup is done.

2. Save the restart files, `restart` and `restart.bak`, from the home directory of the server to tape or to another system.

   Tip: When you have saved these restart files once, they have to be saved again only when the logical volumes owned by the server are changed.

3. The SFS continues to be used. Each time the SFS log file logical volume is filled to a specified level, SFS dumps the state of the log file logical volume to an archive file. Save the log archive files periodically.

Perform this procedure at regular intervals.

If a system failure occurs and you restore the restart files, the recovery procedure depends on whether the log file logical volume is lost. If the log file logical volume is still intact, you can restore the SFS data file logical volume from backup.

This process replays the log records written, because the backup was performed to restore the data volume to the state it was in immediately before the data volume was lost. However, it requires all log archive files written since the data backup to be available.

If the log file logical volume is lost, irrespective of whether the data volume is lost, you must first restore the log file logical volume and then the data volumes from the archive files. You must restore the log file logical volume from the latest available log archive file. This process can also require older log archive files, and restores the log file logical volume to the state it was in when the latest available log archive file was written.

The data volume is then restored in the same way as before, replaying the log records since the backup was taken to restore the data volume to the state it was in when the latest available log archive file was written. This, almost certainly, involves some loss of data.

If you use MRA, make sure to reduce the number of log archive files on the system periodically. These files can be very large, and if they are left to fill up the volume, the SFS cannot archive any data and ends abnormally.

At appropriate intervals, save these files and delete them from the directory. For an SFS named `/.:/cics/sfs/hostB`, the files are written to the `/var/cics_servers/archives/hostB` directory.

Alternatively, you can use the SFS Diagnostic Tool (SDT) to convert your SFS files into flat files, which you can then store on another system. For information about using the SFS diagnostic tool, see the TXSeries documentation library.

Figure 5-2 illustrates how MRA is used in combination with mirroring.

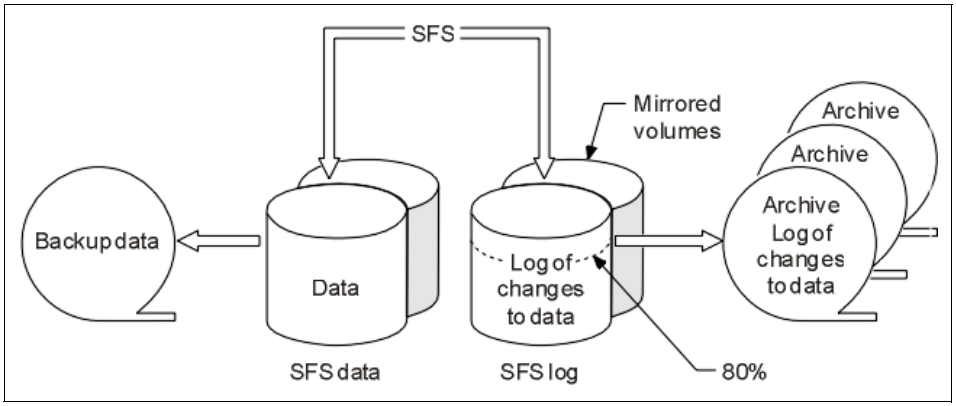


Figure 5-2 MRA used in combination with mirroring for SFS backup

Using the operating system to copy the SFS logical volumes

You can use the operating system to copy the entire contents of a logical volume. For an SFS, copy the restart files, the data volume, and the log volume in a consistent state. The only way you can do this is to copy them simultaneously while the SFS is not running.

The restart files (`restart` and `restart.bak`) are in the home directory of the server. If a system failure occurs, you can restore the SFS data and log volumes from the most current level available and then autostart the SFS.

Enabling and disabling media archiving

Media archiving is enabled and disabled by setting the `MRAArchivingEnabled` attribute in the Structured File Server definitions (SSD). By default, this attribute is set to `no`. Use the following procedure to set the value of the attribute:

1. Shut down the SFS.
2. Use **`cicsupdate`** to change the `MRAArchivingEnabled` SSD attribute to either `yes` or `no`.
3. Perform a warm start of the SFS.

Recovering SFS queues and file management

This section describes how to recover an SFS by moving it to a new machine. It assumes that you have backups of the SFS data file logical volume and the SFS log file logical volume.

Perform the following steps to recover the SFS:

1. Create the SFS to match the original SFS. The SSD for this SFS must exactly match the original SSD of the SFS that you are recovering.
2. Perform a cold start of the SFS.
3. Shut down the SFS.
4. Recover the SFS *log file* logical volume. The method that you use depends on the backups that are available.

   If the SFS is running with MRA enabled, you can recover the log volume by using the log archive files. To do this, you must start the SFS:

   a. Use the TXSeries **sfs** command *without* the **-l** option. Do *not* use the **cicssfs** command.

   b. Then use the **tkadmin recover logvols** command.

   When you use this method, the log volume is recovered to its state when the last log archive file was written. If you used an operating system command to back up the log file logical volume when the SFS was stopped, you can use the copy to recover the logical volume to its state when the copy was taken. For this type of recovery, the SFS must be idle.
5. Recover the SFS *data file* logical volume.

   If you have to recover both the SFS log file and the SFS data file logical volumes, recover the SFS data file logical volume only *after* the log file logical volume is recovered. The method you use depends on which backups are available and how the log file logical volume was recovered:

   – If you performed a backup of the SFS data file logical volume by using the **tkadmin backup lvol** command, you can recover the data by using the **tkadmin recover lvols** command. This recovers the data to a state consistent with the recovered state of the log file logical volume. To do this, the SFS must be started by using the TXSeries **sfs** command, as required for the recovery of the SFS log using MRA.

   – If only the SFS data file requires recovery (that is, if the SFS log file does not require recovery), start the SFS by using the TXSeries **sfs** command *with* the **-l** option, and *not* by using the **cicssfs** command. After recovery, shut down the SFS by using the TXSeries **tkadmin stop server** command.

– If you performed the backup by using an operating system command, you can use the copy of the backup to recover the data file logical volume to its state when the copy was made. However, for this to be a successful recovery, the log file logical volume must be recovered to exactly the same level as the data file logical volume. In practice, this means that copies of both logical volumes must be taken at the same time when the SFS is idle.

6. When both logical volumes are recovered, autostart the SFS in *normal mode* rather than in administrative mode. You can test whether the data has been recovered by opening a list of files on the SFS. For example, if the SFS ID is `hostB`, use the following command:

   ```
   sfsadmin list files -server /.:/cics/sfs/hostB
   ```

   You can also define and start a CICS region, and use it to issue transactions to test the validity of the SFS data.

DB2 file server backup and recovery

There are two types of recovery:

- Restoring to a previous backup (version recovery)
- Restoring to the point of failure

Enabling version recovery

To restore the data to a previous backup, you require a backup of the database that was taken at the same time as a backup of the CICS log. The reason for this is that you have to restore data and the CICS log to the same processing point.

Use the DB2 `BACKUP DATABASE` utility to take offline backups of the DB2 data. Use operating system facilities to take offline backups of the CICS log. You must take backups when the region is shut down and the database is offline. Otherwise, the CICS log and the database can be unsynchronized, and the ability to recover the data is unreliable.

Enabling point of failure recovery

To be able to restore the data and applications to the point of failure, you require a copy of both the database and the CICS log taken when the failure occurred. To enable this, the following conditions must be true:

- You have to mirror the CICS log. The OS documentation describes how to use mirroring to create an offline image backup of an active file system.
- You have to either mirror the logical volumes (LVs) that contain the DB2 data or enable rollforward recovery. The OS documentation describes how to mirror LVs. To use rollforward recovery, you require the following resources:
  – An offline backup copy of the database
  – Rollforward recovery enabled

As described in the **Backup and Recovery** → **Enabling recovery of DB2** → **Point of failure recovery** topic in the TXSeries documentation library, rollforward recovery enables you to build on a restored backup to a specified point in time. Configure the database to enable rollforward recovery.

To do this, set specified parameters in the DB2 database configuration file. For example, use the **`logprimary`** parameter to set the number of primary log files that are created, and use the **`logfilsiz`** parameter to set the number of pages for each log file. We suggest that you mirror the DB2 database, but a lack of coordination is possible between DB2 logging and CICS logging.

DB2 queue and file management recovery procedure

This section describes how to restore a DB2 queue and file management database. It assumes that you have backups of the DB2 data. There are two types of methods that you can use when recovering a DB2 database:

Restore recovery This rebuilds a database to the state that it was in when the backup copy was made. This is also referred to in DB2 as *version recovery*.

Rollforward recovery This builds on the restored copy of the database, making changes to that copy by using the logs containing the modifications made since the last backup.

To recover a DB2 database that is used for CICS queue and file management, perform the following steps:

1. Determine whether you want to restore the data to a previous backup (restore recovery) or to the point at which the failure occurred (rollforward recovery).
2. If you have to re-create the database, verify that the database is configured for CICS queue and file management, as described in the topic about administration of file and queue management on DB2 in the TXSeries documentation library.
3. Start the database. See the TXSeries documentation library (DB2) for instructions about using the `RESTORE DATABASE` utility to restore the database.

> **Tip:** You can use the DB2 `RESTORE DATABASE` utility to create a new database, and then restore the backup image of the data.

Oracle file server backup and recovery

Two types of recovery are available for Oracle file servers:

- Restoring to a previous backup (*version recovery*)
- Restoring to the *point of failure*

Enabling version recovery

To restore the data to a previous backup, you require a backup of the database that was taken at the same time as a backup of the CICS log. You need this backup because you have to restore the data and the CICS log to the same processing point.

Use the Oracle utility for database backup to take offline backups of the Oracle data. Use operating system facilities to take offline backups of the CICS log. Perform both of these backups when the region is shut down and the database is offline. Otherwise, the CICS log and the database can be unsynchronized, and the ability to recover the data is unreliable.

Enabling point of failure recovery

To restore the data and applications to the point of failure, you require a copy of the database and of the CICS log that were taken when the failure occurred. To enable this, the following conditions must be true:

- You have to mirror the CICS log. The operating system documentation describes how to use mirroring to create an offline image backup of an active file system.
- You have to either mirror the logical volumes that contain the Oracle data or enable rollforward recovery. The operating system documentation describes how to mirror logical volumes. To use rollforward recovery, you require the following resources:
  - An offline backup copy of the database
  - Rollforward recovery enabled

As described in the Oracle documentation, rollforward recovery enables you to build on a restored backup to a specified point in time. Configure the database to enable rollforward recovery. See the Oracle documentation for information about how to enable rollforward recovery.

Oracle queue and file management recovery procedure

You can use the Oracle restore utilities to perform a recovery to a particular version, or to perform a point-in-time recovery. See the Oracle documentation for more details.

5.2.3 Configuration backup and recovery

CICS configuration includes three types of information:

- SFS configuration
- CICS region configuration
- Operating system information

Backing up SFS configuration

The configuration and running information for all SFSs on a machine is in the `/var/cics_servers` directory. This directory contains the following subdirectories:

- GSD directory. This contains the gateway server definitions (GSD) autostart and cold start databases for any Peer-to-Peer Communications (PPC) gateway server existing on the machine, and the information that is unique to each gateway.
- SCD directory. This contains the Schema File Definitions (SCD) autostart and cold start databases for all SFSs on the machine.
- SSD directory. This contains the SSD autostart and cold start databases for all SFSs on the machine, and the information unique to each SFS. For each SFS, the SSD directory contains the following files:
  - A lock file that is used to lock a file while it is being updated. The full path name for the lock file is `/var/cics_servers/SSD/cics/sfs/baseName/lock`.
  - A message file that contains the messages logged for the server, with a full path name of `/var/cics_servers/SSD/cics/sfs/baseName/msg`.
  - Two restart files that contain information required by the server for a warm start. The full path names for the restart files are `/var/cics_servers/SSD/cics/sfs/baseName/restart` and `/restart.bak`.

Backing up CICS region configuration

The CICS directory structure for each region is in `/var/cics_regions/<region name>/`, where `<region name>` is the name of the region. This directory structure contains files related to the particular region. The configuration information consists of the following components:

- The permanent database. This contains all of the resource definitions used when the region undergoes a cold start.
- The runtime database. This contains all of the resource definitions used when the region is autostarted.

> **Important:** The permanent and runtime databases do not contain the GSD, SCD, and SSD. These are located on `/var/cics_servers/`.

- The region environment variables.
- The executable files that make up CICS application programs (by default).

  You can copy these files to back up the CICS region. An alternative way is to use the **`cicsexport`** and **`cicsimport`** commands for backup and recovery. This action is useful only for cold starts, because the runtime database is not backed up and restored with these commands.

The **cicsexport** command copies the following data:

– The cold start (permanent) database

  This contains the resource definitions for the region. Suppose that you require any of the region definitions to be shared between the region that you are copying and the region to which you want to restore the copy. Consider maintaining a single physical copy of the definitions and using symbolic links to provide an up-to-date copy in the second region.

– Conversion templates

– All programs in the region's bin directory

– All maps in the region's bin directory

– The contents of the region's data directory

For example, to archive the resource definitions for `regionA` to an output file named `REG1ARCH`, use the following command:

```
cicsexport -r regionA -o REG1ARCH
```

To restore the resource definitions for `regionA`, use the following command:

```
cicsimport -r regionA -i REG1ARCH
```

Before you start using the **cicsexport** and **cicsimport** commands, review the information about them in the TXSeries documentation library.

Backing up CICS operating system information

In this section, we describe techniques to back up CICS information.

Backing up and recovering the subsystem entries

Back up the CICS region subsystem and the SFS subsystem. The subsystem entries for CICS components are updated as a result of an administrative task, and require backup only occasionally. See your operating system documentation for information about where the subsystem entries are located and how to back them up.

Backing up group and user information

Back up the `cics` and `cicsterm` groups. You must also back up the following users:

- cics (the common user for CICS)
- Other customer-defined users related to CICS

The operating system user ID information for CICS components is updated as a result of an administrative task, and requires only occasional backup. See your operating system documentation for information about where the user ID information is located, and about how to back it up.

Backing up environment variables

Back up the CICS environment variables in the operating system-wide context. The OS environment information for CICS components is updated as a result of an administrative task, and requires only occasional backup. See your OS documentation for information about where the environment information is located, and about how to back it up. For the list of environment variables, see the topic about environment variables used by CICS in the TXSeries documentation library.

A full recovery after a complete system failure

A full recovery is required when your CICS system is completely disabled, for example, when the hardware is replaced. This procedure assumes that you have the installation media for TXSeries, DB2 (if you are using DB2 for queue and file management), and that you have the backup tapes that you made earlier.

Perform the following steps:

1. Install TXSeries. If you have to replace the hardware, or if you lose a disk containing the TXSeries, reinstall them in the same way that you originally installed them.
2. Recover queue and file management. Use one of the following procedures, depending on the product that you use:
   - Recover SFS.

     If you have not lost the operating system logical volumes for the SFS, you can autostart the SFS. After you autostart the SFS, look at the `/var/cics_servers/SSD/cics/sfs/baseName/msg` message file.

     A message file created for a normal autostart contains a message similar to the following message:

     ```
     1 A sfs: Initialized ... Thu  Aug 19 07:55:54 1994
     ```

     Use the contents of the message file to diagnose any outstanding problems. For example, a possible problem is “The logical volumes for the SFS no longer exist”.

     For the solution, see “Structured File Server recovery” on page 257, which describes how to recover the SFS logical volumes. If you cannot resolve the errors in the message file, you can attempt to perform a cold start of the SFS. However, use a cold start only as a last resort, because this discards all files that are on the SFS logical volumes.
   - Recover DB2.

     If it is required that you reconfigure the database, see “DB2 file server backup and recovery” on page 262. See the TXSeries documentation library (DB2) for information about restoring the data.

3. Recover CICS service.

   Restore the CICS permanent and runtime resource definition files that are in the `/var/cics_regions/<region name>/database` directory. If a cold start of the region is not required, restore the CICS log files that are in the `/var/cics_regions/<region name>/log` directory.

   If you are able to autostart the SFS, attempt to restart the region normally. If you performed a cold start of the SFS, you must also perform a cold start of the region. However, if you perform a cold start of the region, the runtime resource definition files and the CICS log files are discarded.

   The CICS region reports any problems to its region console file, which is `/var/cics_regions/<region name>/console.nnnnnn`. Review the message files for errors. The `console.nnnnnn` files are generated in sequence. The `console.nnnnnn` file is in the `/var/cics_regions/<region name>/data` directory.

   Every time a new CICS region starts, the number of the `console.nnnnnn` file is incremented by one. The size of this file is controlled through the `MaxConsoleSize` attribute of the RD. If you set `MaxConsoleSize` to 0 (zero), the `console.nnnnnn` file grows until it reaches the limits of your system.

   If you set it to a specific size, the first file is closed, and a new file opens with the next available number. Two messages are issued that tell you when the file is closed and the name of the new file. The **`cicstail`** command automatically continues to tail the new file.

   Tip: If your CICS region is running and the SFS stops for any reason, files are marked as `DISABLED`. If you take administrative action to restore the SFS and the region is still running, you must set the files to `ENABLED`, for example by using the **`CEMT SET ENABLED`** command. If there are many files to reset, it is simpler to restart CICS automatically.

5.3 High-availability solution

Availability requires that the topology must provide some degree of process redundancy to eliminate single points of failure. While vertical scalability can provide this by creating multiple processes, the physical machine then becomes a *single point of failure*. For this reason, a high-availability topology typically involves horizontal scaling for multiple machines.

This section introduces the IBM tool for building a UNIX-based high-availability environment for CICS. For other UNIX platforms, such as Hewlett-Packard UNIX (HP-UX) or Solaris, see platform-specific documentation published by the corresponding vendors.

5.3.1 Overview of IBM High Availability Cluster Multi-processing software and terminology

The IBM tool for building UNIX-based mission-critical computing platforms is the IBM High Availability Cluster Multi-Processing (IBM HACMP™) software. The HACMP software ensures that critical resources, such as applications, are available for processing. It has two major components:

- High availability (HA)
- Cluster multi-processing (CMP)

The primary reason to create HACMP clusters is to provide a highly available environment for mission-critical applications. For example, an HACMP cluster can run a database server program that services client applications. The clients send queries to the server program, which responds to their requests by accessing a database, stored on a shared external disk.

In an HACMP cluster, to ensure the availability of these applications, the applications are put under HACMP control. HACMP takes measures to ensure that the applications remain available to client processes even if a component in a cluster fails. To ensure availability, in case of a component failure, HACMP moves the application (along with resources that ensure access to the application) to another node in the cluster.

In the simplest configuration, one machine is configured to monitor the health (using a “heartbeat”) of another machine. If the second machine fails, resources (such as CICS) are moved over to the first machine to continue availability after a short outage.

The process of the first machine detecting a failure of the second machine, and the applications being moved from the failing machine to the first machine, is called *takeover*. The failed node rejoining the cluster is called *reintegration*.

The following components are the main components of an HACMP cluster:

Nodes These are the machines with failover capabilities. These typically run back-end applications, such as CICS for AIX.

Clients These machines typically run front-end applications (for example, CICS client) that access back-end applications running on nodes. HACMP makes *nodes* highly available, not clients.

Shared external disk devices

These are disks that enable concurrent (multiple nodes can access the disk at the same time) or non-concurrent (only one node can access the disk at a time) access.

Networks Networks are *public* (connects two or more nodes and clients), *private* (node to node, no clients), and *serial* (carries heartbeat traffic, usually an RS-232 cable).

Network adapters The network adapters include the following components:

- Service adapter. An adapter to be used for public network traffic.
- Standby adapter. Backs up a service adapter. A standby adapter is an adapter ready to receive the Transmission Control Protocol/Internet Protocol (TCP/IP) address of a failed service adapter.
- Boot adapter label. This is the TCP/IP address assigned to an adapter after system boot and before HACMP is started and assigns the adapter a service adapter label (address).

For resources to fail over from one node to another, the resource to fail over (disks, volume groups, file systems, network addresses, and application servers) must be defined in a resource group. A resource group is a logical entity that is failed over to another node. Takeover relationships of resource groups exist in the following categories:

Cascading Each node is assigned a priority, and the highest priority node takes over the failing node. When a node of a higher priority reintegrates, it takes over the resource group.

Rotating Each node is assigned a priority and the highest priority node takes over the failing node. When a node of a higher priority reintegrates, no takeover occurs.

Concurrent access Can share resource groups simultaneously.

5.3.2 CICS in the HACMP environment

Every HACMP environment seems to be different. Therefore, it is difficult to develop general cases (for example, always perform *this* procedure to failover CICS, or always do *this* procedure to failover SFS).

Instead, we try to provide some specific examples that address the main points for you to configure a *CICS in HACMP* environment. Although this document addresses CICS, and SFS configurations, it does not include information about other products that might exist in your CICS environment (such as Systems Network Architecture (SNA), DB2, Oracle, and so on).

5.3.3 A demonstration concept of TXSeries in HACMP

This section introduces a demonstration of CICS in an HACMP environment.

Assumptions

This section makes the following assumptions:

- Node 1 is the backup node for node 2. Node 1 is being used by other applications, so failover cannot be disruptive to node 1 users.
- Both node 1 and node 2 have two network interfaces (tr0 and tr1). The tr0 interface is the service adapter, and tr1 is the standby adapter.
- Node 1 is not running SFS, or TXSeries, but the software is installed.
- Node 2 is running SFS and TXSeries.

For the purpose of this example, the TXSeries running on node 2 is called `cicsprod`. The host name for node 1 is `node1`, and the host name for node 2 is `node2`. The SFS name is `/.:/cics/sfs/prod`, and the user name that the SFS runs under is `SFS_SERV`. The SFS data volume name is `sfs_SFS_SERV`.

Note: If you use the default **`cicscp`** command, the SFS name and user name is associated with the `cics` group. You can use this default SFS name. However, there is probably less confusion if you select a more generic name, such as `/.:/cics/sfs/prod1` and `SFS_SERV`.

Figure 5-3 illustrates CICS in an HACMP environment.

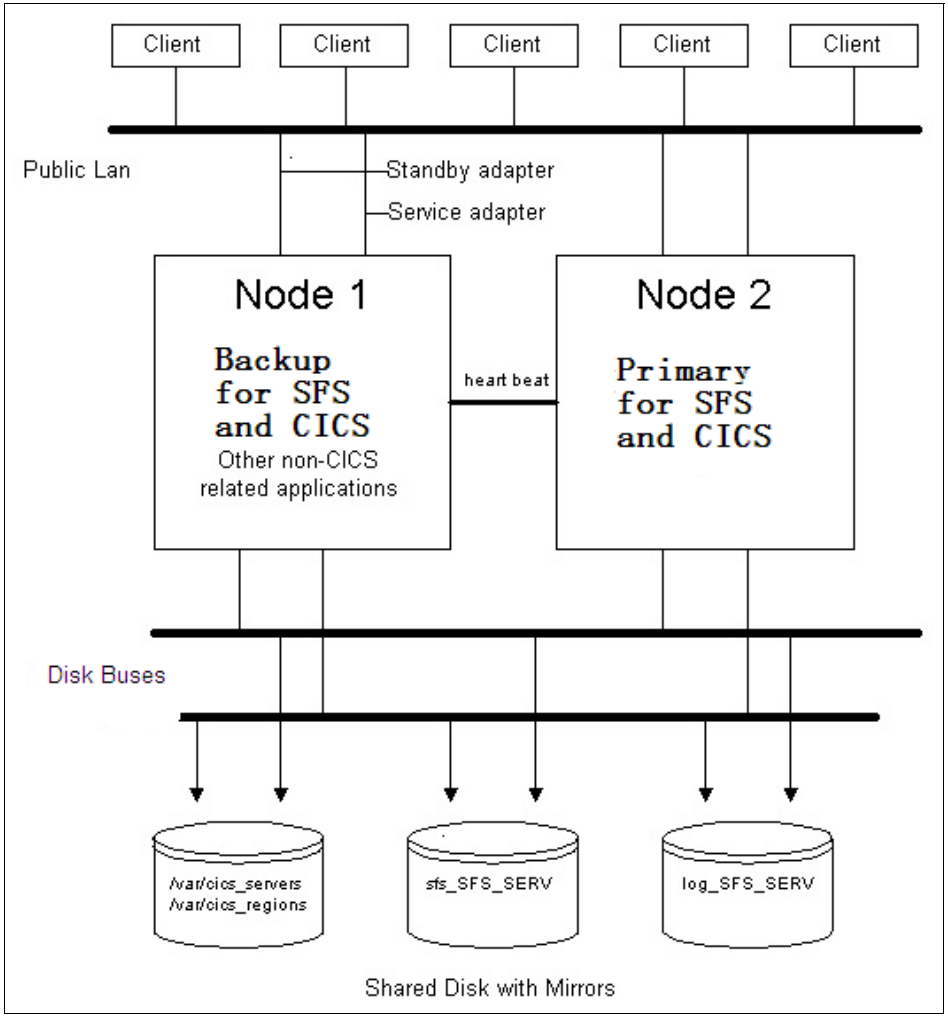


Figure 5-3 CICS in HACMP environment

Install the TXSeries product software and ensure that the group ID for the `cics` and `cicsterm` groups are the same on both machines. Also, ensure that the user ID for the user name `cics` is the same on both machines.

From node 2

Node 2 is the primary node. Perform the following tasks to configure node 2:

1. Configure SFS and TXSeries on node 2 first.
2. To run TXSeries and SFS on both of the nodes, certain file systems and volumes have to be shared. For this demonstration, the easiest way is to have a separate shared file system for `/var/cics_regions`, `/var/cics_servers`, and shared raw logical volumes for the SFS. You can do this now or after you have configured TXSeries and SFS on node 2.
3. Create the SFS with the required name. The user ID for the SFS is `cics`. The SFS user must look the same on both systems, to include the user ID number and the home directory of `/var/cics_regions/SSD/cics/sfs/prod`. Your SFS raw logical volumes have to be on a shared disk accessible to both node 1 and node 2.
4. Create a region. You can create the region in different ways, but you might want to change some parameters in the region definition immediately. For the test case, you have to set `DefaultFileServer=/.:/cics/sfs/prod`. If your SFS data volumes for your SFS are not named `sfs_%S` (*%S* is the short name defined in your SFS stanza definition), you also have to change the volume pointers in the RD stanza for the seven SFS files that the region requires.
5. Ensure that SFS and TXSeries stops and starts.

From node 1

Before any customization is done on node 1, stop TXSeries and SFS on node 2, and failover the file systems and SFS raw logical volumes to node 1. After you do this, perform the following steps:

1. Ensure that the `cics` and `cicsterm` groups have the same group ID as on node 2.
2. Ensure that the `cics` user has the same user ID as on node 2.
3. Define the System Resource Controller entry for the SFS:

   ```
   cicssrccreate -s sfs_server -n /.:/cics/sfs/prod -u SFS_SERV
   ```

4. Ensure that the SFS can be autostarted.
5. Define the SRC entry for the region:

   ```
   cicssrccreate -r cicsprod
   ```

6. Ensure that your TXSeries CICS region starts and stops.

Operation

Whether failed over to node 1 or node 2, the **`cicscp -v start region cicsprod`** command must start the SFS and the region on the currently active node.

5.3.4 TXSeries and high availability: Manual testing

As explained in the previous sections, high availability is one of the most important aspects to evaluate for a TXSeries production environment. The choice of either having or not having an environment HA is *not* transparent to the TXSeries installation.

Therefore, if the idea is to manage the fault tolerance of your transactional environment using a high-availability product (for example, HACMP for AIX), consider making this decision early, and installing and configuring TXSeries accordingly, performing the steps already described.

In this section, we show you how to manually test your TXSeries installation and configuration in terms of high availability as preparation for the automatic failover management that a specific product can supply.

Remember: The configurations of high-availability clusters on distributed platforms are complex, and are documented in the corresponding manuals included with the HA cluster software of each vendor. Because this book is not a manual about how to configure an HA system, we describe the configuration from a TXSeries perspective. We also implement a manual switchover of a failed TXSeries system to simulate an HA cluster environment.

In our demonstration environment, we have two AIX machines sharing some external disks, as shown in Figure 5-4 on page 275:

- PONS1, which is the primary system
- PONS2, which is the backup system

A fastt600 turbo disk is the external shared disk (1722-600) belonging to a storage area network (SAN). For information about the fastt600 turbo disk connection, setup, and configuration, see the SAN-specific documentation.

Figure 5-4 shows the demonstration environment.

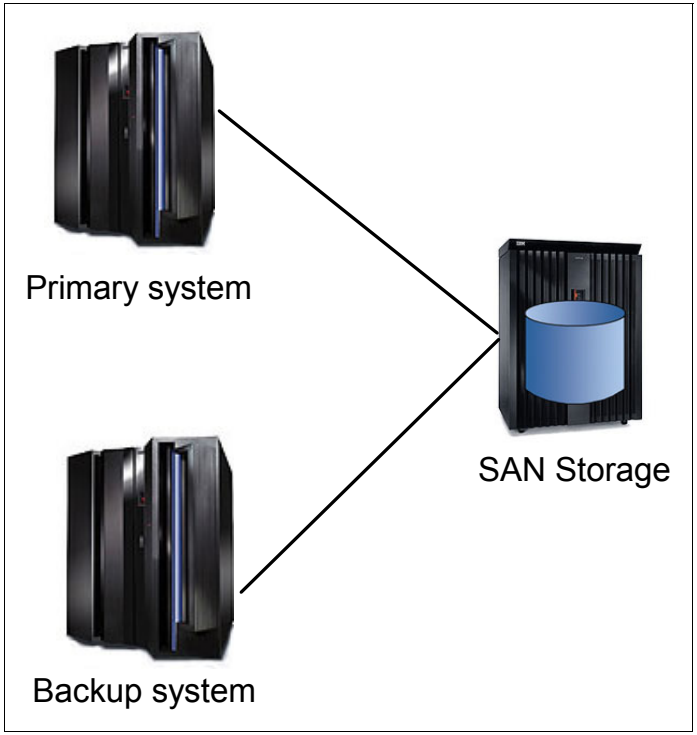


Figure 5-4 System architecture

Preparing the user

Perform the user preparation before the installation of TXSeries on both systems. Define user groups on both systems with the same group identifier:

- cics
- cicsterm

Define users on both systems with the same user identifier:

cics	Member of the user groups `cics` and `cicsterm`.
SSFS_HA	Member of the user group `cics`.

Add user `root` as a member of the `cics` user group in all of the systems.

Preparing the logical volume and file system

Define logical volumes and file systems before the installation of TXSeries. The following tasks are a step-by-step description of the preparation of the test case:

1. We have to identify the shared disk attached to our systems. To do this, on the primary machine and as the root user, run the following AIX command:

   ```
   $ lsdev | grep disk
   ```

   Example 5-1 shows that PONS1 sees the external disk as *hdisk3*.

   Example 5-1 Finding the disks on PONS1

   ```
   {pons1:root}/ -> lsdev | grep disk
   hdisk0       Available 1S-08-00-4,0  16 Bit LVD SCSI Disk Drive
   hdisk1       Available 1S-08-00-5,0  16 Bit LVD SCSI Disk Drive
   hdisk2       Available 1S-08-00-8,0  16 Bit LVD SCSI Disk Drive
   hdisk3       Available 1Z-08-02      1722-600 (600) Disk Array
   Device
   ```

2. We do the same on the backup system, for which we obtain the message shown in Example 5-2. We understand that PONS2 sees the external disk as *hdisk2*.

   Example 5-2 Finding the disks on PONS2

   ```
   {pons2:root}/ -> lsdev | grep disk
   hdisk0      Available 1S-08-00-5,0  16 Bit LVD SCSI Disk Drive
   hdisk1      Available 1S-08-00-8,0  16 Bit LVD SCSI Disk Drive
   hdisk2      Available 1Z-08-02      1722-600 (600) Disk Array
   Device
   ```

3. We create a volume group on the external disk, as shown in Figure 5-5. On the primary system (PONS1), we run the following command:

   `smitty (smit) mkvg`

   > **Requirement:** All of the operations that we have to perform require us to be logged in as the root user.

```
                    Add an Original Volume Group

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

                                                      [Entry Fields]
  VOLUME GROUP name                                  [extvg]
  Physical partition SIZE in megabytes
+
* PHYSICAL VOLUME names                              [hdisk3]
+
  Force the creation of a volume group?               no
+
  Activate volume group AUTOMATICALLY                 no
+
    at system restart?
  Volume Group MAJOR NUMBER                          [60]
+#
  Create VG Concurrent Capable?                       no
+


F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 5-5 Adding a volume group

Here, we provide the name of the new volume group, the disk devices to be included, and the major number to be assigned to it. It is also important to specify that we do not want the volume group activated (varied on) automatically at system restart.

Later, the `varyon` of shared volume groups has to be under the control of the high-availability software (HACMP), so it is coordinated correctly.

4. If the volume group `extvg` contains mirrored disks, turn off `quorum` checking.

   On the command line, we enter **`smitty chvg`** and set quorum checking to `no`, as shown in Figure 5-6.

```
                          Change a Volume Group

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

                                                        [Entry Fields]
* VOLUME GROUP name                                    extvg
* Activate volume group AUTOMATICALLY                  yes                  +
    at system restart?
* A QUORUM of disks required to keep the volume        no                   +
   group on-line ?
  Convert this VG to Concurrent Capable?               no                   +
  Change to big VG format?                             no                   +
  Change to scalable VG format?                        no                   +
  LTG Size in kbytes                                   128                  +
  Set hotspare characteristics                         n                    +
  Set synchronization characteristics of stale         n                    +
   partitions
  Max PPs per VG in kilobytes                          32                   +
  Max Logical Volumes                                  256                  +

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 5-6 Changing the QUORUM

5. Vary on the volume group just created:

   `varyonvg extvg`

6. Before we create any file systems on the shared disk resources, we have to explicitly create the `jfslog` logical volume. We do this so that we can give it a unique name of our own choosing, which is used on all nodes in the cluster to refer to the same log. If we do not do this, it is possible and likely that naming conflicts can arise between nodes in the cluster, depending on what user file systems have already been created.

7. Again we use **smit** to add the `loglvext` log logical volumes for the file systems in the extvg volume group. We enter **smitty mklv**, and select the volume group `extvg` to which we add the first new `jfslog` logical volume, as shown in Figure 5-7.

```
                          Add a Logical Volume

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

[TOP]                                                   [Entry Fields]
  Logical volume NAME                                  [loglvext]
* VOLUME GROUP name                                     extvg
* Number of LOGICAL PARTITIONS                         [1]                   #
  PHYSICAL VOLUME names                                [hdisk3]             +
  Logical volume TYPE                                  [jfslog]             +
  POSITION on physical volume                           middle              +
  RANGE of physical volumes                             minimum             +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                   []                    #
    to use for allocation
  Number of COPIES of each logical                      2                   +
    partition
  Mirror Write Consistency?                             active              +
  Allocate each logical partition copy                  yes                 +
[MORE...12]

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 5-7 Adding a logical volume for log

8. Ensure that the `Logical volume TYPE` is set to `jfslog`, and the `Number of COPIES of each logical partition` is set to `2`. After we create the `jfslog` logical volume, we have to ensure that we format the log logical volume with the following command:

   ```
   logform /dev/loglvext
   ```

   Answer yes (`y`) to the prompt about whether to destroy the old version of the log.

9. At this point, we are ready to create the two logical volumes used by SFS, as shown in Figure 5-8. Assuming that our SFS user ID is `cics`, the logical volumes for the server are `log_SSFS_HA` and `sfs_SSFS_HA`.

```
                              Add a Logical Volume

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

[TOP]                                                   [Entry Fields]
  Logical volume NAME                                 [log_SSFS_HA]
* VOLUME GROUP name                                    extvg
* Number of LOGICAL PARTITIONS                        [10]                  #
  PHYSICAL VOLUME names                               [hdisk3]             +
  Logical volume TYPE                                 []                   +
  POSITION on physical volume                          middle              +
  RANGE of physical volumes                            minimum             +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                  []                    #
    to use for allocation
  Number of COPIES of each logical                     1                   +
    partition
  Mirror Write Consistency?                            active              +
  Allocate each logical partition copy                 yes                 +
[MORE...12]

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 5-8 Adding a logical volume for SFS

10. The operation for adding the `log_SSFS_HA` logical volume is the same as step 9.
11. Next, change the access permission of these logical volumes:

    ```
    chown cics:cics /dev/*log_SSFS_HA
    chown cics:cics /dev/*sfs_SSFS_HA
    ```

12. Create two logical volumes on which to mount the two file systems that are used by TXSeries:
    - The logical volume `sfsvar` for the file system `/var/cics_servers`
    - The logical volume `cicsvar` for the file system `/var/cics_regions`
13. Repeat the logical volume creation steps for both `sfsvar` and `cicsvar`.

14. Create the file systems on the logical volumes that we have just defined. In the command line, enter the following fast path:

    `smitty crjfslv`

    The two file systems are configured, as shown in Figure 5-9.

```
                  Add a Standard Journaled File System

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

                                                        [Entry Fields]
* LOGICAL VOLUME name                                   sfsvar              +
* MOUNT POINT                                           [/var/cics_servers]
  Mount AUTOMATICALLY at system restart?                no                  +
  PERMISSIONS                                           read/write          +
  Mount OPTIONS                                         []                  +
  Start Disk Accounting?                                no                  +
  Fragment Size (bytes)                                 4096                +
  Number of bytes per inode                             4096                +
  Allocation Group Size (MBytes)                        8                   +
  Logical Volume for Log                                                    +





F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 5-9 Adding a file system for /var/cics_servers

15. The operation to add a file system for `/var/cics_regions` is the same as step 12 on page 280. Remember to set the access permission of the two mount point to user cics and user group cics with the following commands:

    - `chown cics:cics /var/cics_regions`
    - `chown cics:cics /var/cics_servers`
    - `chmod 775 /var/cics_regions`
    - `chmod 775 /var/cics_servers`

Information: Consider the following points when configuring the logical volume and file system:

- In the case of a Telnet server configuration, remember to create an additional logical volume on the shared disks for the file system `/var/cics_clients`.
- If you are using HP-UX or Sun Solaris systems that use the file system `/var/cics_src` as the subdirectory in which to save a database with dynamic information about the status of CICS regions and servers (SFS and PPC Gateway servers), remember to define a specific logical volume on the shared disks for it.
- Apply the same method to all of the directories for CORE dumps, system dumps, and transaction dumps if you choose not to save them under the default location `/var/cics_regions`.

Example 5-3 shows the output of the **`lsvg -l extvg`** command on the primary system.

Example 5-3 Logical volume configuration result

```
{pons1:root}/ -> lsvg -l extvg
extvg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT
POINT
loglvext            jfslog     1     1     1    closed/syncd    N/A
log_SSFS_HA         jfs        32    32    1    closed/syncd    N/A
sfs_SSFS_HA         jfs        32    32    1    closed/syncd    N/A
sfsvar              jfs        32    32    1    closed/syncd
/var/cics_servers
cicsvar             jfs        10    10    1    closed/syncd
/var/cics_regions
```

Creating the region and the SFS

After the installation of TXSeries on both systems, configure SFS and CICS regions on both the systems.

Creating the SFS and region on PONS1

Create SFS and region on PONS1, as shown in Example 5-4.

Example 5-4 Creating SFS and region on PONS1

```
#cicscp -v create sfs_server /.:/cics/sfs/SFS_SHA
#cicscp -v start sfs_server /.:/cics/sfs/SFS_HA StartType=cold
#cicscp -v create region CICSHA \
DefaultFileServer="/.:/cics/sfs/SFS_HA"
#cicscp -v start region CICSHA StartType=cold
```

Check the status of the region and SFS using the following command:

```
lssrc -a | grep cics
```

Creating the configuration on PONS2

To create the configuration on PONS2, perform the following steps:

1. Stop the region and SFS on PONS1:

   ```
   cicscp -v stop region CICSHA
   cicscp -v stop sfs_server /.:/cics/sfs/SFS_HA
   ```

2. Unmount the `/var/cics_regions` and `/var/cics_servers` file systems on PONS1 using the **`unmount`** command:

   ```
   umount /var/cics_regions
   umount /var/cics_servers
   ```

3. Vary off the `extvg` volume group on PONS1 using the **`varyoff`** command:

   ```
   varyoffvg extvg
   ```

4. Import the `extvg` volume group on PONS2, as shown in Figure 5-10:

   ```
   smitty importvg
   ```

```
                          Import a Volume Group

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

                                                      [Entry Fields]
  VOLUME GROUP name                                 [extvg]
* PHYSICAL VOLUME name                              [hdisk2]             +
  Volume Group MAJOR NUMBER                         [60]                 +#









F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure 5-10 Importing the volume group

5. Vary on the `extvg` volume group on PONS2 using the **`varyon`** command. Mount the `/var/cics_regions` and `/var/cics_servers` file systems using the **`mount`** command:

   ```
   varyon extvg
   mount /var/cics_regions
   mount /var/cics_servers
   ```

6. Define the SRC entry for the SFS on PONS2:

   ```
   cicssrccreate -s sfs_server -n /.:/cics/sfs/SFS_HA -u SSFS_HA
   ```

7. Ensure that the SFS can be automatically started:

   ```
   cicscp -v start sfs_server /.:/cics/sfs/SFS_HA
   ```

8. Define the SRC entry for the region:

   ```
   cicssrccreate -r CICSHA
   ```

9. Ensure that the CICS region can be automatically started:

   ```
   cicscp -v start region CICSHA
   ```

Verifying the failover of TXSeries systems

To verify the failover, perform the steps described in “Creating the configuration on PONS2” on page 283. However, do not perform steps 6 on page 284 and 8 on page 284, because the SFS and the CICS region definitions are already created.

Information: For more information about high availability, see the *Preparing TXSeries systems for high availability on AIX* white paper.

5.4 User exits

A user exit (also referred to as a *user exit point*) is the point in a CICS program at which CICS can transfer control to a program that you have written (a *user exit program*), and can then resume control when your program has finished its work.

You do not have to use any of the user exits, but they are useful if you want to extend and customize the functionality of your CICS system to meet your specific requirements. This section describes how to implement a user exit with an example of a distributed program link (DPL) user exit.

5.4.1 User exits principle

A CICS region invokes a particular user exit if the user defined the exit in the region. See 5.4.3, “Implementing user exits” on page 287 for more information about how to define the user exit.

Region invoking user exits

User exits are programs with an entry of a particular format routine. The routine must be defined, as shown in Example 5-5.

Example 5-5 Definition of routine cics_UE_entry

```
cics_UE_Return_t cics_UE_entry( cics_UE_Header_t *UE_Header,
cics_UE_Specific_t *UE_Specific)
```

A CICS region transfers parameters to the `UE_Header` and `UE_Specific` when it invokes the program. See the TXSeries documentation library for a more detailed definition of the two structures, `cics_UE_Header_t` and `cics_UE_Specific_t`, and also the routine. Users can customize this routine and write their own logic in the routine. They can obtain information related to the region, transaction, and runtime instances through the input parameters.

Types of user exits

TXSeries supports approximately ten user exits, as shown in Table 5-1. Each user exit is identified with a number. See the TXSeries documentation library for a more detailed description of these user exits.

Table 5-1 CICS-supported user exits

User exit number	User exit name	Description
13	UE015013	Program autoinstall
15	UE014015	Task termination
17	UE052017	Dump request
25	UE014025	Dynamic transaction routing
31	UEI09031	Front-end programming interface (FEPI) data conversion
33	UE046033	Dynamic resource definition
50	UE015050	Dynamic DPL
51	UE016051	Syncpoint
52	UE115052	Internet Inter-Object Request Broker (ORB) Protocol (IIOP) security exit
53	UE115053	IIOP certificate exit

5.4.2 User exit chaining

TXSeries V8.1 extends the user exit facility to define multiple user programs for a given user exit. If your application architecture includes Workload Manager (WLM) and your application needs a DPL user exit as well, the user exit chaining facility helps you to achieve it. This facility also helps if multiple products want to extend the same user exit.

You can also define the sequence of the user programs execution associated for a given user exit. The `UserExitSequence` attribute in `PD.stanza` determines the order of execution.

5.4.3 Implementing user exits

TXSeries CICS user exits are also CICS programs. The following sections describe how to implement a user exit for dynamic DPL.

Coding the user exits

TXSeries CICS user exits can only be written in C language. TXSeries provides a sample user exit program:

`/usr/lpp/cics/samples/userexit/cicsuxit.c`

The user can use this source code for all user exits. The entry routine, `cics_UE_entry`, is defined, as shown in Example 5-6.

Example 5-6 CICS user exit entry routine

```
cics_UE_Return_t cics_UE_entry( cics_UE_Header_t *UE_Header,
cics_UE_Specific_t *UE_Specific)
{
cics_UE_Return_t wott;
strcpy(UE_Header->UE_Workarea,UGH);
switch(UE_Header->UE_Number)
{
/*
* Task end user exit UE014015
*/
case 15:
strcpy(UE_Header->UE_Workarea,CASE15);
wott=cics_UE014015(UE_Header, (cics_UE014015_t *)
UE_Specific);
return(wott);
break;
/*
* Dump request user exit UE052017
*/
case 17:
strcpy(UE_Header->UE_Workarea,CASE17);
UE_Header->UE_TraceOnExit=UE_Yes;
return(UE_Bypass);
break;
/*
* Dynamic transaction routing user exit UE014025
*/
case 25:
strcpy(UE_Header->UE_Workarea,CASE25);
wott=cics_UE014025(UE_Header, (cics_UE014025_t *)
```

```
UE_Specific);
return(wott);
break;
/*
* FEPI SEND/RECEIVE data conversion exit UE109031
*/
case 31:
strcpy(UE_Header->UE_Workarea,CASE31);
wott=cics_UE109031(UE_Header, (cics_UE109031_t *)
UE_Specific);
return(wott);
break;
/*
* Dynamic Resource Definition user exit UE046033
*/
case 33:
strcpy(UE_Header->UE_Workarea,CASE33);
wott=cics_UE046033(UE_Header, (cics_UE046033_t *)
UE_Specific);
return(wott);
break;
/*
* Dynamic DPL user exit UE015050
*/
case 50:
strcpy(UE_Header->UE_Workarea,CASE50);
wott=cics_UE015050(UE_Header, (cics_UE015050_t *)
UE_Specific);
return(wott);
break;
/*
* Syncpoint user exit UE016051
*/
case 51:
strcpy(UE_Header->UE_Workarea,CASE51);
wott=cics_UE016051(UE_Header, (cics_UE016051_t *)
UE_Specific);
return(wott);
break;
/*
* IIOP security user exit UE115052
*/
case 52:
/* Note: we could be called from multiple threads here! */
strcpy(UE_Header->UE_Workarea,CASE52);
```

```
wott=cics_UE115052(UE_Header, (cics_UE115052_t *)
UE_Specific);
return(wott);
break;
/*
* IIOP SSL certificate user exit UE115053
*/
case 53:
/* Note: we could be called from multiple threads here! */
strcpy(UE_Header->UE_Workarea,CASE53);
wott=cics_UE115053(UE_Header, (cics_UE115053_t *)
UE_Specific);
return(wott);
break;
case 13:
strcpy(UE_Header->UE_Workarea,CASE13);
wott=cics_UE015013(UE_Header, (cics_UE015013_t *)
UE_Specific);
return(wott);
break;
default:
strcpy(UE_Header->UE_Workarea,DEFAULT);
UE_Header->UE_TraceOnExit=UE_No;
return(UE_Normal);
break;
}
}
```

CICS transfers control to the `cics_UE_entry` routine when an exit is invoked. Based on the user exit number as indicated in the `UE_Header->UE_Number`, the user exit program calls the corresponding user exit logic.

In the DPL user exit, the routine `cics_UE015050` is invoked (Example 5-7).

Example 5-7 Distributed program linked user exit routine

```
cics_UE_Return_t cics_UE015050(
cics_UE_Header_t *UE_Header,
cics_UE015050_t *UE_Specific)
{
strcpy(UE_Header->UE_Workarea,LINK);
UE_Header->UE_TraceOnExit=UE_Yes;
switch (UE_Specific->UE_Dplfunc)
{
case (UE_LINKSEL) : strcpy(UE_Header->UE_Workarea,LINKSEL);
break;

case (UE_LINKUNKNOWN) :
strcpy(UE_Header->UE_Workarea,LINKUNKN);
break;
case (UE_LINKTERM) :
strcpy(UE_Header->UE_Workarea,LINKTERM);
break;
case (UE_LINKABEND) :
strcpy(UE_Header->UE_Workarea,LINKABEND);
break;
}
return(UE_Normal);
}
```

The `UE_Specific` structure contains user exit-specific information, such as DPL exit structure, and contains information about whether the user exit is invoked by a DPL `LINK` call, or by a DPL `LINK` abnormal end of task (abend). You can customize or add your own logic in this routine. For example, you can log a message onto your application log when a DPL abnormally ends or stops.

Building the user exits

Because TXSeries CICS User exits can only be written in C, to compile a user exit, you must have a TXSeries-supported C compiler. Use the following command to compile the user exit programs:

```
#xlc_r -c -I/usr/lpp/cics/include cicsuxit.c
#xlc_r -e cics_UE_entry -o cicsuxit cicsuxit.o –liconv
```

The simplest way is to modify the `cicsuxit.c` and use the TXSeries makefile:

```
#cd /usr/lpp/cics/samples/userexit (On AIX)
#make
```

Deploying the user exits

To deploy a user exit is almost the same as deploying a TXSeries CICS program. The only difference is the `UserExitNumber` attribute in the program definitions (PD). For a particular user exit, you have to set the corresponding user exit number to the `UserExitNumber` attribute.

You have to put the user exit module into a path that TXSeries CICS regions can access. Typically, we put it under the region's `bin` directory.

The following command defines a user exit in a region:

```
cicsadd -c pd -r <region name> CICSUXIT PathName="cicsuxit"
UserExitNumber=50
```

The user exits must be loaded into the application servers (CICS application server processes) to take effect. Therefore, restart the region to enable the CICS application server processes to load the user exit.

Typical uses of user exits

User exits are typically used for the following purposes:

User Exit 17	To suppress dumps based on a certain application condition.
User Exit 25 and 50	To perform intelligent routing when a DPL or dynamic transaction routing (DTR) request is received.
User Exit 15	To perform application cleanups during abnormal or normal termination of tasks.

5.5 Monitoring and statistics

TXSeries provides statistics, monitoring, and tracing tools that you can use to gain information about running a region. You can use the offline utility programs `cicssfmt`, `cicsmfmt`, and `cicstfmt` to format the information provided by these tools. You can then analyze the formatted data to provide information about any resource contention or other problems that can affect the performance of your CICS system.

Most of the information related to monitoring and statistics is already described in CICS manuals. See the TXSeries documentation library for more information about how to use these facilities. This section introduces some monitoring and statistics scenarios with these techniques in real cases.

5.5.1 CICS statistics

Example 5-8 shows the first panel of an output of the **`cicssfmt statsfile`** command.

Example 5-8 Statsfile output

```
Statistics Filename: statsfile
Date: Wed Nov 16 11:01:10 2005                                Page: 1

                    CICS STATISTICS INTERVAL SUMMARY REPORT


Type: INTERVAL Statistics Collection
Time Start : Mon Nov 14 15:00:00 2005

DUMP          # WRITEERR      # DUMPSWRITTEN
              0               0

FILE          # Tot. READS    # Tot. WRITES   # Tot. BROWSES  # Tot. DELETES
              # Tot. UPDATES  # Tot. OPENS    # Tot. CLOSES
              0               0               0               0
              0               0               0
```

Important: The first time stamp is not the time that the statistic report was generated, it is the time that the file is formatted. You can find this time stamp on every page of this formatted output.

The CICS statistics file contains the following categories of information:

- DUMP (dump statistics). Records the number of dumps written, and the number of write errors that occurred while dumps were taken.
- FILE (file statistics). Records input and output activity of application requests against the files.
- INTERSYSTEM (intersystem communication (ISC) statistics). Records ISC statistics.
- JOURNAL (journal statistics). Provides statistical information for every journal defined to a region.
- PROGRAM (program statistics). Provides statistical information for all programs defined in the PD.
- RUNTIMEDB (runtime database statistics). Records statistics on the usage of the runtime database and the collision characteristics of the supporting hash table.

- STORAGE (storage statistics). Provides information about the region pool, task-shared pool, and task-private pool.
- TASK (task statistics). Provides information relating to tasks.
- TDQUEUE (transient data statistics). Provides statistical information about each transient data queue (TDQ) defined in the TDD. It also accumulates statistics for all transient data requests since region startup.
- TERMINAL (terminal statistics). Provides information about terminals defined in WD. This information is important to show how users are loading the system.
- TRANSACTION (transaction statistics). Provides information about all of the transactions defined in the Terminal Definition (TD).
- TSQUEUE (temporary storage statistics). Provides information about temporary storage queues (TSQ) defined in the TSD.
- UNITOFWORK (LUW statistics). Provides accumulated statistics relating to LUWs.

Each category contains a specific index of statistics details, such as the FILE section in Example 5-8 on page 292. You can find the list of the detail index in the topic about the statistic records in the TXSeries documentation library.

The interval statistics are collected every 3 hours, and the end-of-day statistics are collected at 24:00 every day. All the statistics are sequentially put into the file. From these details, you can track the running status of the region.

For example, we can track the task-shared pool usage of a particular region, as shown in Example 5-9.

Example 5-9 CICS task-shared pool usage statistics

```
Statistics Filename: statsfile
Date: Wed Nov 16 11:01:10 2005
Page: 70

                                    CICS STATISTICS INTERVAL SUMMARY REPORT

STOR                  TASK SHARED POOL SIZE                TASK SHARED POOL THRESH

                            CURRENT TBYTES ALLOCATED       # TBYTES HIGHWATER
                            # TBYTES ALLOCATED                     # TBYTES FREED
                            # TALLOCS SUCCEEDED                  # TALLOCS FAILED
                            # TFREES SUCCEEDED                    # TFREES FAILED
                            # TSHORT OF STORAGE                   MAX. ADDR THASH
                            WORST THASH BUCKET
                            1048576
943718
                             520
7998
                            0
0
                            0
0
                            0
0
                            0
1
                             279
```

With this information, we can draw a chart for the task-shared pool usage of the region, as shown in Figure 5-11.

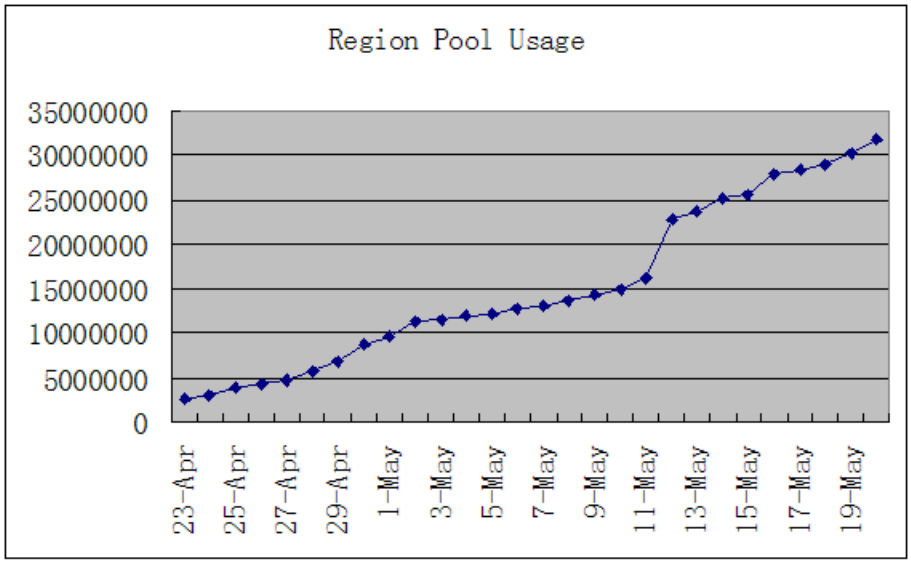


Figure 5-11 Task-shared pool usage

With this method, you can also monitor usage information of other resources, such as frequency of ISC, TSQ usages, and even a particular program's hit rate.

CICS Statistics Display

TXSeries also provides an online transaction utility, CICS Statistics Display (CSTD), for displaying the statistics information for different CICS resources:

- Temporary storage and transient data
- Files
- Terminals
- Transactions
- Programs
- Intersystem communication
- Tasks
- Storage pools

Using CSTD, you can monitor transaction and program execution rates, the number of times the application servers are full, the peak number of tasks run, and many other region-related tunable information. To gather this information, the CSTD transaction utility uses the **`EXEC CICS STATASTICS`** command. CSTD is a pseudo-conversational transaction. Running this transaction does not block any CICS application server (CICSAS) process.

You can also use the auto update facility to monitor CICS resource usage continuously over a period of time. The auto refresh facility updates the resource statistics every 60 seconds by default. You can also change the auto refresh display interval by choosing option 12 from the CSTD transaction menu. When you run CSTD transaction from a 3270 client to start the online CICS statistics display, the following panel is displayed, as shown in Figure 5-12.

```
CSTD   08/10/14 15:02:47       CICS Statistics Display                DFHCST.00
                                                                       V01.00


     Option ===>


         1 - TS and TD Statistics
                                                 2 - Pool Storage Statistics
         3 - Miscellaneous Statistics
                                                 4 - ISC Details Statistics
         5 - File Statistics
                                                 6 - Terminal Statistics
         7 - Program Statistics
                                                 8 - Transaction Statistics
         9 - Class Max Task Statistics
                                                10 - ISC Summary Statistics
        11 - Transaction/Program Rates
                                                12 - Set/Display Statistics



 Enter option and press enter
PF1=Help              Enter=Refresh              PF3=Exit           Clear=Exit
```

Figure 5-12 CICS Statistics Display Panel

The following list provides more information about CSTD:

- If you enter 1 and press Enter, the CICS Statistics Display shows the current TSQ and TDQ statistics. You can alternatively use the **CST1** transaction.

  Using these statistics, you can monitor if there are any errors while accessing the queues. You can also monitor how many auxiliary TSQ operations were performed since the last interval of collected statistics. It is always advised to use the main TSQ in applications, rather than the auxiliary TSQs, to achieve better performance.

  Figure 5-13 on page 298 shows typical output of the **CST1** transaction:

  – The Last Reset field shows the time since the statistics were collected.

  – The Main field shows the Main TSQ operations count for Reads, Writes, and Deletes.

  – The Aux field shows Auxiliary TSQ operations count like Reads, Writes, and Deletes.

- The Abort field shows the number of times that the TSQ operations rolled back. If you see a high count, analyze the console and symrecs log files.
- The Commits field indicates the number of transaction commit requests received by the TSQ.
- The Exhausted field indicates the number of times that the TSQ operation has encountered a `NOSPACE` condition. If the memory is frequently exhausted, the task-shared pool size can be increased if the main TSQ is used. If SFS is used as the File Manager, SFS data and log volume sizes can be increased. You can set `TSQAgeLimit` in the RD stanza to inform CICS to clean up unused TSQs.
- The Peak Num field indicates the largest concurrent number of TSQs existing. If main memory queues are mostly used, you need to consider increasing `MaxTaskSharedPool` in the RD stanza. The Peak Num field can help in tuning `MaxTaskSharedPool`.
- The Current Num field indicates the number of TSQs existing in the region at this instance.
- The Longest Queue field indicates the largest number of items that has existed in any TSQ.
- The I/O Errors field indicates the number of times the TSQ has encountered the `IOERR` condition. This can occur if the temporary storage is full. This can also occur if File Manager is not reachable. Analyze console log messages for further action.
- The Auth Errors field indicates the number of TSQ requests made without sufficient authority. Verify console log messages and provide necessary authority for the respected user before rerunning the transaction.
- The Aged field indicates the number of TSQs that have been deleted by the CICS-private Temporary Storage Queue Aging transaction (CAGE). The time elapsed since they were last accessed was greater than the TS queue aging limit. You can set the `TSQAgeLimit` in the RD stanza to inform CICS to clean up unused TSQs.
- The CAGE field indicates the number times that the CAGE has been run.
- The Del Conflict field indicates the number times a TSQ has encountered another user of a queue when deleting it, and had to wait before retrying.
- The Remote Reqs field indicates the number of TSQ requests to queues on remote CICS regions.
- The Extra field indicates the TDQ global statistics count for extrapartition queue reads and writes. Monitor disk space if many writes are performed.
- The Intra field indicates the TDQ global statistics count for intrapartition queue reads and writes.

– The Remote field indicates the TDQ global statistics count for remote queue reads and writes.

– The Trigger field indicates the TDQ global statistics count for triggerable queue reads.

Figure 5-13 shows typical **CST1** output.

```
CST1   08/10/14 15:29:36       CICS Statistics Display              DFHCST.01

15:00:00 :Last Reset
                              Temporary Storage
                        Reads       Writes      Deletes
                 Main:  0000000073  0000001206  0000000031
                 Aux:   0001499289  0001499291  0001499289
           Aborts:        0000000000         I/O Errors:            0000000000
           Commits:       0000000001         Auth Errors:           0000000000
           Exhausted:     0000000000         # Aged:                0000000000
           Peak Num:      0000000048         # CAGE:                0000000000
           Current Num:   0000000002         Del Conflict:          0000000000
           Longest Queue: 0000000001         Remote Reqs:           0000000000

                             Transient Data
                              Reads       Writes
                    Extra:   0000000000  0003378814
                    Intra:   0000000000  0000000000
                    Remote:  0000000000  0000000000
                    Trigger: 0000000000  0000000000


Enter=Refresh PF3=Main Menu                    PF6=Start Autoupdate Clear=Exit
```

Figure 5-13 Sample output of TSQ and TDQ statistics from CST1 transaction

- If you enter 2 and press Enter, the CICS Statistics Display shows the current Memory statistics. You can alternatively use the **CST2** transaction. Figure 5-14 on page 299 shows typical output of the **CST2** transaction:

  – Region Pool, Task Shared Pool, and Task Private Pool allocation request count and deallocation request count are displayed in **CST2**. If you find more allocation requests than deallocation requests, there is a possibility of a memory leak in the respective pool of memory.

  – The output also shows the configured size, Threshold size, In use size and Peak usage of the memory pools.

  – If the peak usage (High field) reaches the Threshold value, you have to consider increasing the respective memory pool parameter in `RD.stanza`. Increasing the memory pool requires restarting the region. When the usage crosses the Threshold value, warning messages are logged in the region console file.

- Region pool memory is extensively used for CICS internal purposes. The number of active CICSAS processes and resource definitions in the region is directly proportional to the usage of the region pool.
- The Task Shared pool is used for storing the Main memory temporary storage queues, COMMAREA data shared across programs, and the **GETMAIN** command with the **SHARED** option. If the Task Shared pool is continuously increasing over a period of time, some tasks might be leaking. You can find the leaking tasks by enabling the storage fields in CICS monitoring facility (CMF).
- The Task-Private Storage Abends field indicates how many task-private storage abend conditions have been raised during the selected period. You should check the CSMT log and transaction storage abend statistics to find which transactions are being abnormally ended because of task-private storage problems.
- The Tasks Using Task-Private field indicates how many CICS tasks were using task-private storage during the selected period.

Figure 5-14 shows typical output of **CST2**.

```
CST2   08/10/14 15:30:09       CICS Statistics Display              DFHCST.02

15:00:00 :Last Reset
                            Memory Pool Statistics

              Size       Threshold  In Use     High       Allocated  Returned
Region Pool:
              0020485760 0018437184 0009069296 0009566016 1919767008 1915279200
Task Shared:
              0001048576 0000943718 0000452948 0000516351 0044578880 0044126452
                      Allocations             Deallocations
                  Good        Bad          Good        Bad          SOS
  Region Pool:
                  0236298437  0000000000   0236292922  0000000000   0000000000
  Task Shared:
                  0000379068  0000000000   0000378065  0000000000   0000000000
  Task-Private:
                  0075564336  0000000000   0075563462  0000000000

            Task-Private Storage Abends:                      0000000000
            Tasks Using Task-Private:                         0000000107


Enter=Refresh PF3=Main Menu                     PF6=Start Autoupdate Clear=Exit
```

Figure 5-14 Sample output of Memory Pool statistics from CST2 transaction

- If you enter `3` and press Enter, the CICS Statistics Display shows miscellaneous statistics related to different CICS resources and functionalities. You can alternatively use the **`CST3`** transaction. Figure 5-15 on page 301 shows typical output of the **`CST3`** transaction:
  - The Num Trans Started field indicates the total number of transactions run in the region during the selected period. This field is useful to identify the workload periods on the system.
  - The Trans Not Started field indicates the total number of transactions for the region that CICS was unable to start during the selected period. Analyze the console and CSMT log files to identify the reason.
  - The Times AppServs Full field indicates the total number of times that the number of active transactions reached the maximum number of application servers for the region, during the selected period. Check if any transactions are taking a long time to complete or are HUNG.
  - The Peak Number Tasks field indicates the maximum number of task requests that the region was processing, at any one moment in time, during the selected interval.

    This includes both queued and running tasks requests. If the number shown is well in excess of the `MaxServer` attribute in the region definition file, and the Times AppServs Full statistic indicates that the `MaxServer` limit is reached regularly, you should consider increasing the `MaxServer` value.
  - The Total field in Abends section indicates the total number of transactions abnormally terminated in the region, during the selected period. Analyze the console, CSMT, and symrec files for identifying the reason.
  - The Addressing field in Abends section indicates the total number of exceptions raised in the region, due to invalid memory addresses, during the selected period.
  - The Protection field in Abends section indicates the total number of exceptions raised in the region, due to protection violations, during the selected period.
  - The Op Sys field in Abends section indicates the total number of exceptions raised in the region, due to operating system-detected errors, during the selected period.
  - The ICE chain maximum Length field indicates the maximum number of interval control elements (ICEs) that have existed in the chain to date.
  - The ICE chain current Length field indicates the current number of ICEs in the internal chain.
  - The Total ENQ requests field indicates the total number of distinct resources that have been enqueued with the **`EXEC CICS ENQ`** command to date.

- The ENQ requests that waited field indicates the number of **EXEC CICS ENQ** requests that waited until the specified resource was available.
- The LOCK requests that waited field indicates the number of internal lock requests that waited until the specified resource was available.
- The Dumps Written field indicates the number of dumps that were written.
- The Dump Write Errors field indicates the number of write errors that occurred while dumps were being taken. Write errors can be caused due to authorization issues, or the file system is full.

Figure 5-15 shows typical output of **CST3**.

```
CST3   08/10/14 15:06:34       CICS Statistics Display              DFHCST.03

15:00:00 :Last Reset        Miscellaneous Statistics

   Num Trans Started:                        0006621787
   Trans Not Started:                        0000000000
   Times AppServs Full                       0004278948
   Peak Number Tasks:                        0000001000
                                                                Abends:
                                                   Total:            0000000000
                                                   Addressing:       0000000000
                                                   Protection:       0000000000
                                                   Op Sys:           0000000000
   ICE chain maximum Length:                 0000000009
   ICE chain current Length:                 0000000008
   Total ENQ requests:                       0000443979
   ENQ requests that waited                  0000217927
   LOCK requests that waited:                0000000000
                                             Dumps Written:          0000000000
                                             Dump Write Errors:      0000000000


Enter=Refresh PF3=Main Menu                     PF6=Start Autoupdate Clear=Exit
```

Figure 5-15 Sample output of miscellaneous statistics from CST3 transaction

- If you enter 4 and press Enter, the CICS Statistics Display shows intersystem communication statistics. It provides detailed information about all of the ISC facilities supported by TXSeries. You can alternatively use **CST4**. Figure 5-16 on page 303 shows typical output of the **CST4** transaction:
  - The File Control field indicates the number of file control (FC) function shipping requests that were received from remote systems, sent to and received from remote systems, and forwarded to remote systems.
  - The Transient Data field indicates the number of Transient Data queue function shipping requests that were received from remote systems, sent to and received from remote systems, and forwarded to remote systems.

- The Temporary Storage field indicates the number of temporary storage (TS) queue function shipping requests that were received from remote systems, sent to and received from remote systems, and forwarded, to remote systems.
- The Interval Control field indicates the number of interval control (IC) function shipping requests that were received from remote systems, sent to and received from remote systems, and forwarded to remote systems.
- The DPL field indicates the number of DPL function shipping requests that were received from remote systems, sent to and received from remote systems, and forwarded to remote systems.
- The Transaction Route field indicates the number of transaction routing channel requests that were received from remote systems, sent to and received from remote systems, and forwarded to the application-owning region (AOR).
- The Terminal Defs field indicates the number of terminal definitions that were received from the terminal-owning region (TOR) and sent to the AOR.
- The DTP Conversations field indicates the number of DTP conversations that have been allocated.
- The Max Queued field indicates the maximum number of function shipping requests that were locally queued.
- The Curr Queued field indicates the number of function shipping requests that are currently queued. If the value reports a continuously high number, the `TCPProcessCount` value in the LD stanza can be increased for `CICS_TCP` communication. For `ppc_tcp` communication, tune the `RPCListenerCount` value in the RD stanza to increase the Remote Procedure Call (RPC) processing count. If not resolved, enable CMF to spot the issue.
- The Sent field indicates the number of function shipping requests that have been sent to the remote system.
- The Purged field indicates the number of locally queued function shipping requests that have been purged from the queue.
- The Send attempts field indicates the number of times that a locally queued function shipping request has been made to the remote system.

Figure 5-16 shows typical output of **CST4**.

```
CST4   08/10/14 15:07:10       CICS Statistics Display              DFHCST.04

15:00:00 :Last Reset
                       Intersystems Communications Stats for: @11T
                                                   Page 0001 Total:   0002

                                  In         Out        Forwarded
              File Control:       0000000000 0000000000 0000000000
              Transient Data:     0000000000 0000000000 0000000000
              Temporary Storage:  0000000000 0000000000 0000000000
              Interval Control:   0000000000 0000000000 0000000000
              DPL:                0000000000 0000000000 0000000000
              Transaction Route:  0000000000 0000000000 0000000000
              Terminal Defs:      0000000000 0000000000
              DTP Conversations:             0000000000
                              Queued Function Ships
                          Max Queued:    0000000000
                          Curr Queued:   0000000000
                          Sent:          0000000000
                          Purged:        0000000000
                          Send attempts: 0000000000


Enter=Refresh    PF3=Main Menu                      PF8=Forward    Clear=Exit
```

Figure 5-16 Sample output of Intersystems Communications statistics from the CST4 transaction

- If you enter 5 and press Enter, the CICS Statistics Display shows File statistics. It provides detailed information about all of the operations performed on all of the files used by TXSeries region:
  - Reads
  - Writes
  - Browses
  - Deletes
  - Updates
  - Opens
  - Closes

  You can alternatively use the **CST5** transaction.

Figure 5-17 shows typical output of the **CST5** transaction.

```
CST5   08/10/14 15:07:50       CICS Statistics Display              DFHCST.05

00:00:00 :Last Reset
                           File Statistics  (Item 0000 Total:   0000 )
Name                Writes                Deletes               Opens
         Reads                  Browses               Updates            Closes









                                     Totals:
0000000  0000000000 0000000000 0000000000 0000000000 0000000000 0000000 0000000
 ===> No Files selected <===
                Display inactive files: N
                                        Start display with file:

Enter=Refresh     PF3=Main Menu                      PF8=Forward     Clear=Exit
```

Figure 5-17 Sample output of File statistics from CST5 transaction

- To achieve better performance for more frequently used file operations, you can tune the region by setting the following parameters:
  - You can enable the caching facility by setting the environment variable for more frequently used files.
  - Set the **Recoverable** attribute of the FD stanza to `non-recoverable` for read-only files.
  - If you are using SFS as the File Manager, make the following adjustments:
    - Set the **ErrorIsolation** attribute in the FD stanza for recoverable files to `off`.
    - Use different SFS for storing files.
    - Tune the **BufferPoolSize** and **OpThreadPoolSize** parameters in `SSD.stanza`.
  - If you are using DB2 as the File Manager, make the following adjustments:
    - Rebind the TXSeries-supplied DB2 packages with the `REOPT ALWAYS` option.
    - If TASK A does a **READ UPDATE** followed by **REWRITE** and TASK B does **READ** only concurrently on the same DATASET, then the **READ** of TASK B waits for TASK A to do a **SYNCPOINT**. The wait on Task B can be avoided by setting the **CICSDB2CONF_BIND_UR** environment variable to `1`.

- If you enter `6` and press Enter, the CICS Statistics Display will show terminal statistics. It provides detailed information about all of the operations performed on terminals. You can alternatively use the **CST6** transaction. Figure 5-18 shows typical output of the **CST6** transaction:
  - The Name field indicates the name of the terminal.
  - The Inputs field indicates the number of messages received from the terminal.
  - The Outputs field indicates the number of messages received by the terminal.
  - The Transmit field indicates the number of transmission failures between CICS and CICS terminal.
  - The TTI field indicates the number of terminal transaction initiation (TTI) transactions that were unsuccessfully submitted against the terminal.
  - The Transactions field indicates the number of TTI transactions submitted against the terminal.
  - You can enter a terminal name in the Start display with terminal field to display the terminal statistics from the terminal name that you entered.
- You can set the `RecoverTerminal` attribute to `no` in the Terminal Definition stanza file if terminal recovery is not required. This improves the performance on terminal operations.

```
CST6   08/10/14 15:08:40       CICS Statistics Display              DFHCST.06

15:00:00 :Last Reset
                         Terminal Statistics  (Item 0001 Total:   1001 )
                                              Errors
 Name       Inputs        Outputs      Transmit     TTI           Transactions
 A019       0000000000    0000007811   0000000000   0000000000    0000006251
 A119       0000000000    0000007840   0000000000   0000000000    0000006272
 A219       0000000000    0000007808   0000000000   0000000000    0000006250
 A319       0000000000    0000007842   0000000000   0000000000    0000006271
 A359       0000000000    0000007781   0000000000   0000000000    0000006226
 A419       0000000000    0000007809   0000000000   0000000000    0000006251
 A519       0000000000    0000007832   0000000000   0000000000    0000006266
 A619       0000000000    0000007813   0000000000   0000000000    0000006248
 A719       0000000000    0000007785   0000000000   0000000000    0000006226
 A859       0000000000    0000007782   0000000000   0000000000    0000006225
                                      Totals:
 0001001    0000000000    0007803925   0000000000   0000000000    0006243316

              Display inactive terminals: N
                                      Start display with terminal:

Enter=Refresh     PF3=Main Menu                      PF8=Forward     Clear=Exit
```

Figure 5-18 Sample output of terminal statistics from the CST6 transaction

- If you enter 7 and press Enter, the CICS Statistics Display shows program statistics. It provides the number of times that each program got run in the region. You can alternatively use the **CST7** transaction. Figure 5-19 shows typical output of the **CST7** transaction.

  You can enter a program name in the Start display with program field to display the program statistics from the program name that you entered.

- The CICSAS process loads the application program to the process heap for execution and unload after execution completes. From program statistics, if you find any program accesses frequently, you can set the `Resident` attribute in PD stanza to enable caching for that program (except for MF COBOL and Java programs). The number of programs cached by each CICSAS process is decided by the `ProgramCacheSize` attribute setting in the RD stanza.

```
CST7   08/10/14 15:23:45       CICS Statistics Display              DFHCST.07

15:00:00 :Last Reset
                        Program Statistics  (Item 0001   Total:  0082  )
 Name                       Name                          Name
           # Run                        # Run                          # Run
 CO100000  0000363830       CO110000  0000363718        CO120000  0000363718
 CO130000  0000363718       CO140000  0000363718        CO150000  0000000000
 CO160000  0000363761       COB10000  0000363756        COB20000  0000363756
 COB30000  0000363756       COB40000  0000363756        COB80000  0000363783
 COB90000  0000363778       COLN0000  0000363670        COM10000  0000363738
 COM20000  0000363738       COM30000  0000363738        COM40000  0000363738
 CONT0000  0000363692       CONT2     0000363670        DFHCACQX  0000000000
 DFHCADB   0000000000       DFHCADB0  0000000000        DFHCALF   0000000000
 DFHCALF0  0000000000       DFHCCINX  0000000000        DFHCDCN   0000000000
 DFHCDCN0  0000000000       DFHCEBR   0000000000        DFHCEBR0  0000000000
      Number Programs:      0000082
                                                 Total Programs Run: 0011642774

             Display inactive programs: N
                                   Start display with program:

Enter=Refresh     PF3=Main Menu                      PF8=Forward     Clear=Exit
```

Figure 5-19 Sample output of Program statistics from CST7 transaction

- If you enter 8 and press Enter, the CICS Statistics Display shows transaction statistics. It provides the number of times that each transaction ran in the region. You can alternatively use the **CST8** transaction. Figure 5-20 shows typical output of the **CST8** transaction:
  - The S Abends field indicates the number of abends raised during transaction execution related to task-private pool storage. If the abend count is nonzero, verify your application for task-private pool usage, for example, using the **GETMAIN** storage application programming interface (API). Also verify the `MaxTaskPrivatePool` attribute setting in RD stanza.
  - You can enter a transaction name in the Start display with transaction field to display the transaction statistics from the transaction name that you entered.
  - The St Abends field indicates the total number of storage abends raised for all transactions.
  - The Total Transaction field indicates the total number of transactions run in the region during the selected period.
  - The Run field indicates the total number of transactions instances run in the region during the selected period.

```
CST8   08/10/14 15:26:32       CICS Statistics Display                DFHCST.08

15:00:00 :Last Reset
                            Transaction Statistics  (Item 0001 Total:   0093 )
      # Run                       # Run                       # Run
 Name            S Abends   Name            S Abends   Name            S Abends
 CACQ 0000000000 0000000    CADB 0000000000 0000000    CAGE 0000000000 0000000
 CAIN 0000000000 0000000    CALF 0000000000 0000000    CARP 0000000000 0000000
 CATS 0000000000 0000000    CAUT 0000000000 0000000    CCIN 0000000000 0000000
 CDCN 0000000000 0000000    CEBR 0000000000 0000000    CECI 0000000000 0000000
 CECS 0000000000 0000000    CEDF 0000000000 0000000    CEMT 0000000000 0000000
 CESF 0000000000 0000000    CESL 0000000000 0000000    CESN 0000000000 0000000
 CFTS 0000000017 0000000    CGWY 0000000000 0000000    CHAT 0000000001 0000000
 CICE 0000000017 0000000    CIOD 0000000000 0000000    CIOF 0000000000 0000000
 CJDB 0000000000 0000000    CKQC 0000000000 0000000    CKTI 0000000000 0000000
 CLAM 0000000017 0000000    CLU0 0000000000 0000000    CMBT 0000000000 0000000
 Total Transaction:           0000000093
            # Run:           0007383921         St Abends:         0000000000

            Display inactive transactions: N
                                   Start display with transaction:

Enter=Refresh     PF3=Main Menu                      PF8=Forward     Clear=Exit
```

Figure 5-20 Sample output of Transaction statistics from CST8 transaction

- If you enter `9` and press Enter, the CICS Statistics Display shows TranClass statistics. You can alternatively use the **`CST9`** transaction. Figure 5-21 on page 309 shows typical output of the **`CST9`** transaction. With IBM TXSeries for Multiplatforms V8.1, the maximum number of transaction types (`TCLASS`) is no longer restricted to 10:
  - The Tcl field indicates the name of the TranClass.
  - The Max Activ field indicates the value defined in the MaxActive attribute of the TranClass Definition (TCD) stanza file for the respective TranClass.
  - The Queue Len field indicates the value of the `PurgeThresh` attribute of the TCD stanza file for the class. When the queued transaction limit crosses the value of the `PurgeThresh` attribute, the transactions will be purged:
    - If the value is `0`, the size of the queue is unlimited.
    - If the value is `1`, no transactions are allowed to queue.
    - If the value is *`n`*, the number of transactions that are allowed to queue is *`n-1`*.
  - The Current field indicates the total number of Class tasks in the region during the selected period. This includes both queued and running tasks.
  - The Peak field indicates the maximum number of task requests for Class that the region was processing at any one moment in time, during the selected interval. This includes both queued and running task requests.
  - The At Max field indicates the number of times that starting a new task resulted in the number of Class running tasks in the region reaching the maximum value during the selected interval. The maximum value is defined by the `MaxActive` attribute of the TCD file. If the number indicates that the maximum is reached at regular intervals, you should consider your use of transaction classes for the respective TranClass.

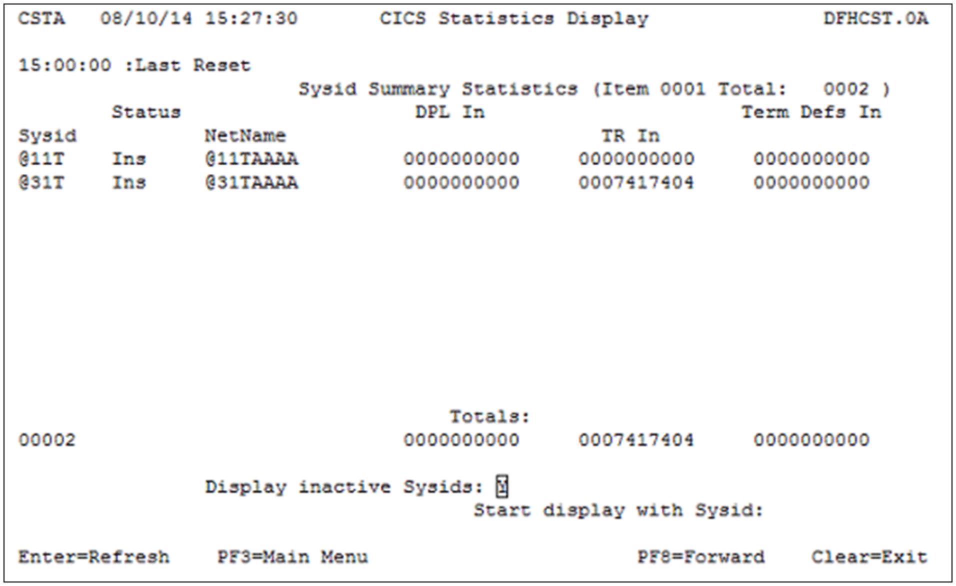
    If this number, and its equivalent for the other transaction classes, indicates that the maximum is rarely reached, but at the same time the statistic `Times AppServs Full` in the CST3 panel indicates that the `MaxServer` limit is reached regularly, you should consider increasing the `MaxServer` attribute of the region definition file. However, this condition could also be due to a high number of task requests with no Class.
  - The Rejects field indicates the number of times a Class task was rejected because the region had already reached the maximum number of running and queued tasks allowed for this Class, during the selected interval. The maximum value is defined by the `MaxActive` and `PurgeThresh` attributes of the TCD file. If this value is high, it might be that the `PurgeThresh` attribute has been set too low in the TCD file.

Figure 5-21 shows typical output of **CST9**.

```
CST9   08/10/14 15:27:00       CICS Statistics Display              DFHCST.09

15:00:00 :Last Reset
         TranClass Statistics  (Item   0001        Total:    0010        )

     -------------------------  Class Max Task -------------------------
         Max Activ             Current                At Max
  Tcl                Queue Len             Peak                   Rejects
DFHTCL01 0000000001 0000000000 0000000000 0000000000 0000000000 0000000000
DFHTCL02 0000000001 0000000000 0000000000 0000000000 0000000000 0000000000
DFHTCL03 0000000001 0000000000 0000000000 0000000000 0000000000 0000000000
DFHTCL04 0000000001 0000000000 0000000000 0000000000 0000000000 0000000000
DFHTCL05 0000000001 0000000000 0000000000 0000000000 0000000000 0000000000
DFHTCL06 0000000001 0000000000 0000000000 0000000000 0000000000 0000000000
DFHTCL07 0000000001 0000000000 0000000000 0000000000 0000000000 0000000000
DFHTCL08 0000000001 0000000000 0000000000 0000000000 0000000000 0000000000
DFHTCL09 0000000001 0000000000 0000000000 0000000000 0000000000 0000000000
DFHTCL10 0000000001 0000000000 0000000000 0000000000 0000000000 0000000000



Enter=Refresh     PF3=Main Menu                      PF8=Forward     Clear=Exit
```

Figure 5-21 Sample output of TranClass statistics from CST9 transaction

- If you enter `10` and press Enter, the CICS Statistics Display shows ISC Summary statistics. You can alternatively use the **CSTA** transaction. Figure 5-22 shows typical output of the **CSTA** transaction:
  - The Sysid field indicates the local name for a remote system with which the CICS region will communicate.
  - The Status field indicates connection status with a remote system.
  - The NetName field indicates the name of the network to which the remote system is attached.
  - The DPL In field indicates the number of DPL function shipping requests that were received from remote systems.
  - The TR In field indicates the number of Transaction Routing requests that were received from the TOR.
  - The Term Defs In field indicates the number of terminal definitions that were received from the TOR.
  - The Totals field indicates the total number of counts for Sysid, DPL In, TR In, and Term Defs In requests.
  - You can enter a Sysid name in the Start display with Sysid field to display the Sysid summary statistics from the Sysid name that you entered.



```
CSTA   08/10/14 15:27:30       CICS Statistics Display          DFHCST.0A

15:00:00 :Last Reset
                        Sysid Summary Statistics (Item 0001 Total:   0002 )
        Status                    DPL In                      Term Defs In
Sysid           NetName                         TR In
@11T    Ins     @11TAAAA         0000000000     0000000000     0000000000
@31T    Ins     @31TAAAA         0000000000     0007417404     0000000000









                                     Totals:
00002                            0000000000     0007417404     0000000000

                Display inactive Sysids: N
                                       Start display with Sysid:

Enter=Refresh    PF3=Main Menu                        PF8=Forward    Clear=Exit
```

Figure 5-22 Sample output of ISC Summary statistics from CSTA transaction

- If you enter 11 and press Enter, the CICS Statistics Display shows Program/Transaction Rates statistics. You can alternatively use the **CSTB** transaction. Figure 5-23 shows typical output of the **CSTB** transaction:
  - The Running field in Tasks section indicates the total number of all user tasks that are currently running.
  - The Dispatchable field in Tasks section indicates the total number of all user tasks that are currently dispatchable (ready to run).
  - The Run field and the per sec field in the Transactions section display the total number of transactions run and the number of transactions run per second since the last statistics reset or the last panel refresh.
  - The Run field and the per sec field in the Programs section display the total number of Programs run and the number programs run per second since the last statistics reset or the last panel refresh. You can use this information during tuning of the TXSeries region.

```
CSTB   08/10/14 15:28:08       CICS Statistics Display              DFHCST.0B


                    Program/Transaction Rates                  reg81

       Tasks:  Running:      000100    Dispatchable:     000556



      Since Last Statistics Reset at 15:00:00
                         Transactions              Programs
                       Run       per sec         Run       per sec
                    0007443453    409.62      0011907551     54.23


      Since Last Screen Refresh at --------
                         Transactions              Programs
                       Run       per sec         Run       per sec
                    ----------    ------      ----------    ------


 Press enter or PF6 to calculate since last refresh

Enter=Refresh PF3=Main Menu                     PF6=Start Autoupdate Clear=Exit
```

Figure 5-23 Sample output of Program/Transaction Rates statistics from CSTB transaction

- If you enter 12 and press Enter, the CICS Statistics Display shows Statistics Display/Set statistics. You can alternatively use the **CSTZ** transaction. Figure 5-24 shows typical output of the **CSTZ** transaction. You can set different statistic collection settings:
  - The Last Reset field indicates the reset time for the last statistics counters.
  - The Next Reset field indicates the time at which the next statistics counters will be reset.
  - The Interval field indicates the statistics collection reset interval. The default value is 3 hours. You can change it based on your requirements.
  - The End-of-Day field indicates the time at which end-of-day statistics should be collected. The default value is `00:00:00`.
  - You can also clear statistics counters if you want to retest the scenarios without restarting the region.
  - Using the Interval for Auto Refresh displays of transaction field, you can set the auto refresh interval to continuously monitor different statistics. The default value is 60 seconds. This value is used if you choose the Auto refresh option in any of the statistics transaction outputs.

```
CSTZ   08/10/14 15:28:42       CICS Statistics Display                DFHCST.0Z


                           Statistics Display/Set

           Clear Statistics Counters N  (Y or N)
                                      Current counters will be sent to file
  15:00:00 :Last Reset
  18:00:00 :Next Reset
             Interval                          End-of-day
               Hours   : 03  (00-24)             Hours   : 00  (00-23)
               Minutes: 00  (00-59)              Minutes: 00  (00-59)
               Seconds: 00  (00-59)              Seconds: 00  (00-59)



  Recording:       Y  (Y or N)
                        interval/unsolicited statistics

  Interval for Auto Refresh displays of transaction:
                                                 060  (15-999) (seconds)

 Overtype values and press Enter
Enter=Action                     PF3=Main Menu                      Clear=Exit
```

Figure 5-24 Sample output of Statistics Display/Set statistics from CSTZ transaction

Requirement: To obtain statistic information for a particular program or transaction, you must use the **`cicssfmt i statsfile`** command to get a detailed statistic report.

5.5.2 CICS monitoring facility

This section describes the steps to enable CMF, and how to collect the monitor information from the output file.

Defining a transient data queue

All of the monitoring information is logged on to a TDQ. Therefore, you must define a TDQ first. Example 5-10 shows an applicable TDQ for a monitor definition (MD). For demonstration purposes, we define the TDQ as an extrapartition TDQ. The output file containing the monitoring information can be found in `/var/cics_regions/<region name>/data/MQONQ.out`. Example 5-10 shows the command and the sample TDQ definition file.

> **Tip:** Set the TDQ **`RecordType`** parameter to `variable_length` for monitoring:
>
> ```
> cicsadd -c tdd -r txdemo MONQ RecordType=variable_length
> ```

Example 5-10 TDQ for an MD

```
MONQ:

GroupName         = ""
ActivateOnStartup = yes
ResourceDescription = ""
AmendCounter       = 0
Permanent         = yes
IOMode            = output
WhenOpened        = at_startup
OpenMode          = truncate
RecordType        = variable_length
RecordTerminator = 0
IndirectQueueId = ""
FacilityType      = file
RecoveryType      = none
TriggerLevel       = 0
RemoteSysId       = ""
RSLKey            = public
FacilityId        = ""
RecordLen         = 0
TriggeredTransId = ""
RemoteName        = ""
DestType          = extrapartition
ExtrapartitionFile = "MONQ.out"
TemplateDefined = no
```

Configuring the monitor definition stanza

Defining an MD is not accomplished in the same way as defining other CICS resources. It is not necessary to add a new item into the MD resource database. We have to modify the default item in `MD.stanza` using the **`cicsupdate`** command. Example 5-11 shows the command.

Example 5-11 Sample MD configuration

```
cicsupdate -c md -r txdemo MonitorStatus=yes TDQ=”MONQ”
```

Example 5-12 shows the sample definition of the MD stanza.

Example 5-12 Sample definition of the MD stanza

```
:ml10
ResourceDescription="Monitor Definition"
AmendCounter=1
Modifiable=no
Conversational=no
MonitorStatus=yes
TDQ="MONQ"
Exclude=""
Include=""
UserMonitorModule=""
```

After you configure the MD, perform a cold start of the region. The monitoring data is written to `/var/cics_regions/<region name>/data/MONQ.out`.

Formatting the monitoring output

The monitoring information output in `MONQ.out` is not in a readable format. You require a tool to read and format the monitoring information into the format you want. TXSeries provides a sample tool, `cicsmfmt`, to format the monitor output.

But `cicsmfmt` can only print out part of the information in the output file. See the TXSeries documentation library for details about the routines used to format monitor output. You can write your own tool to format the monitor information by referring to the supplied sample program `cicsmfmt.c`.

5.5.3 Programming to collect monitoring and statistics information

In addition to the monitoring information and statistics provided in previous sections, users might want some other information, such as answers to the following questions:

- Which programs are running in the region at a particular time?
- What is the running status of every task?
- What is the statistic information at a particular time?

This section describes how to get this information.

Collecting statistics information

To get statistics information, you can use the **COLLECT STATISTICS CICS** command. You can find details about this command in the TXSeries documentation library. The following example shows the API format:

```
EXEC CICS COLLECT STATISTICS SET(ptr-ref) LASTRESETHRS(data-area)
LASTRESETMIN(data-area) LASTRESETSEC(data-area) ......
```

It is easy to get the `LASTRESETHRS`, `LASTRESETMIN`, and `LASTRESETSEC`. The following sections describe how to parse the content pointed to by `ptr-ref` with a sample of storage statistics.

Structures used in the sample

The following structures are in this sample application:

- **struct CICSSTAT_RecordList** (Example 5-13). This is the structure that the `ptr-ref` points to. The definition is in `/usr/lpp/cics/include/cics_stat.h`.

Example 5-13 struct CICSSTAT_RecordList

```
struct CICSSTAT_RecordList
{
cics_ulong_t NumRecords;
CICSSTAT_Record_t Records[1]; /* used as variable size */
};
```

Example 5-13 on page 315 includes the following variables:

NumRecords This indicates how many statistics records there are in the Records array.

Records This is an array of statistics records. Each record contains either summary or resource-specific statistics for a requested statistics class. The Records field is a variable-sized field, as described in the comment, rather than a single member array.

It only points to the address of the first `CICSSTAT_Record_t` member in the array. CICS defines the structure this way because the `CICSSTAT_Record_t` structure is also a structure of variable size.

- **`CICSSTAT_Record_t`** (Example 5-14). This is the structure of the data records written to the output medium for later processing as required.

Example 5-14 Definition of CICSSTAT_Record_t

```
struct CICSSTAT_Record
{
cics_ulong_t Id;
CICSSTAT_Mesg_t Category;
cics_char_t ItemName[CICSSTAT_ITEM_NAME_LEN + 1];
cics_ushort_t NumStats;
CICSSTAT_Detail_t Details[1]; /* used as variable size */
};
typedef struct CICSSTAT_Record CICSSTAT_Record_t;
```

Example 5-14 includes the following variables:

Id This identifies which set of statistics this statistic belongs to, for example, `transactions` or `temporary storage`. The definition of Id is in the chapter about statistic records in the TXSeries documentation library.

Category This is the key to the statistics message catalog, which identifies the piece of text that can be used to describe this statistics set.

ItemName This is an optional field that can be used to store a name if the statistics are the details for a particular item rather than a general category. For example, the statistics record for temporary storage queue `TS1` stores `TS1` in `ItemName`.

NumStats This is how many items there are in the Details array.

Details This is the array of statistics for this record. For example, for a temporary storage queue, there might be one element in the array for the number of times the queue was read, one for the number of times it was written, one for the number of times a record was deleted, and one for the maximum number of records in the queue.

- **`CICSSTAT_Detail`** (Example 5-15). This structure is used as part of the CICSSTAT_Record.

Example 5-15 Definition of CICSSTAT_Detail_t

```
struct CICSSTAT_Detail
{
cics_ulong_t Id;
CICSSTAT_Mesg_t Description;
cics_slong_t Value;
};
typedef struct CICSSTAT_Detail CICSSTAT_Detail_t;
```

Example 5-15 includes the following variables:

Id This indicates the detail category of this statistic detail record. For example, maximum region pool size or current in used region pool size. You can find the definition of Id in the chapter about statistic records in the TXSeries documentation library.

Description This describes the statistic record.

Value This indicates the value corresponding to the Id.

Parsing the statistics

The structure sequence returned by `ptr-ref` is shown in Figure 5-25.

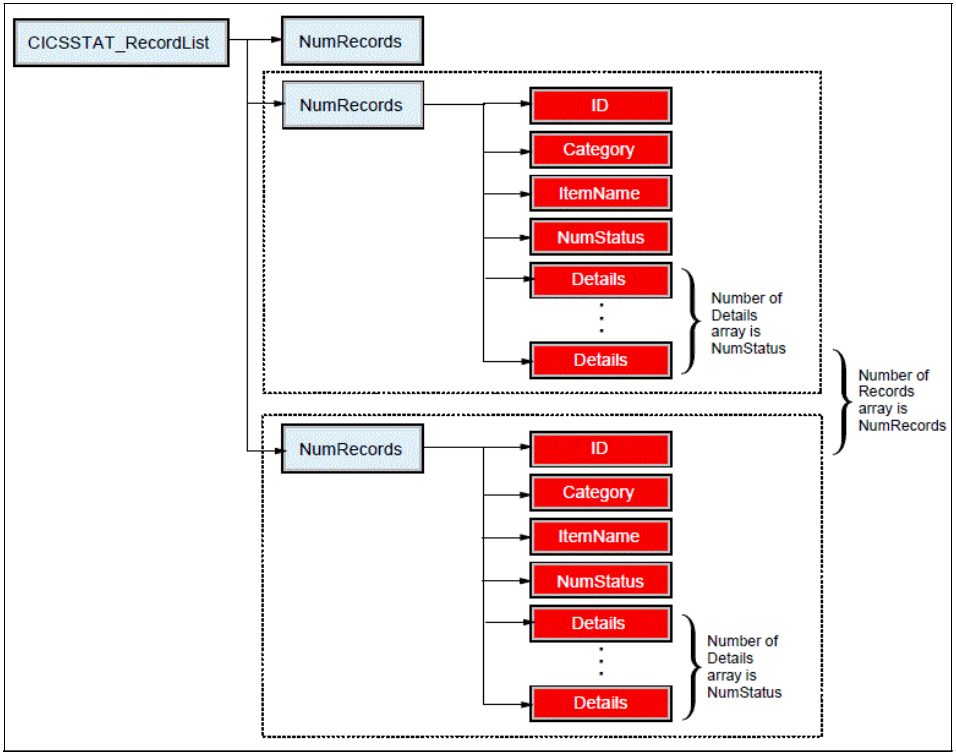


Figure 5-25 Structure of the statistics returned from COLLECT STATISTICS

The source code `monitor.ccs` provided as part of this book shows how to parse the statistics in the routines **`ParseRecordList()`** and **`GetStorageStatDetail()`**. This sample program only collects parts of the information from the storage-related statistics.

Collecting region information

The `monitor.ccs` sample program shows how to collect system information and task information with CICS API.

Collecting system information

Use the **`INQUIRE SYSTEM`** CICS API to collect system information. See the TXSeries documentation library for details about the use of this command.

The `monitor.ccs` sample program shows how to collect the region name and region system ID with the following API, as shown in Example 5-16.

Example 5-16

```
CICSSTAT_RecordList NumRecords
. . . .
ID
Category
ItemName
NumStatus
Details
Details
NumRecords
. . . .
ID
Category
ItemName
NumStatus
Details
Details
NumRecords
Number of
Details
array is
NumStatus
Number of
Details
array is
NumStatus
Number of
Records
array is
NumRecords
```

Collecting task information

Use the **`INQUIRE TASKLIST`** and **`INQUIRE TASK`** CICS APIs to collect task information. See the TXSeries documentation library for details about the use of these commands.

The `monitor.ccs` sample program shows how to collect the task list, and some parameters of a particular task, with these APIs as a sample.

Sample program

The `monitor.ccs` sample program shows you how to use CICS APIs to collect statistics and monitoring information.

Resources required by the sample program

This program requires a TDQ for output. Example 5-17 shows the TDQ definition.

Example 5-17 TDQ for the monitor.ccs sample program

```
PMON:
GroupName=""
ActivateOnStartup=yes
ResourceDescription="CICS Abnormal Conditions"
AmendCounter=0
Permanent=yes
RemoteSysId=""
RemoteName=""
RSLKey=public
DestType=extrapartition
IOMode=output
ExtrapartitionFile="PMON.out"
WhenOpened=at_startup
OpenMode=truncate
RecordType=line_oriented
RecordLen=1024
RecordTerminator=0
IndirectQueueId=""
FacilityType=file
RecoveryType=none
TriggeredTransId=""
TriggerLevel=0
FacilityId=""
TemplateDefined=no
```

You can customize your own output device by modifying the **`printlog()`** routine.

Compiling the sample program

Compile the program with the following command:

```
#cicstcl -lC monitor.ccs
```

Deploying the sample program

To deploy the program, complete the following steps:

1. Add a PD with the command shown in Example 5-18.

Example 5-18 Adding a PD for the sample program

```
#cp monitor /var/cics_regions/<region name>/bin
#cicsadd -c pd -r <region name> -B MONITOR \
PathName="monitor" RSLKey=public
#cicsadd -c td -r <region name> -B PMON \
RSLKey=public RSLCheck=none ProgName="MONITOR"
```

2. Perform a cold start of the region.

Starting the program

Users can start the performance monitor (PMON) transaction in a cicsterm, or start the transaction when the region starts. After it is called for the first time, the transaction is called every 30 minutes. If you do not want this transaction to be called every 30 seconds, you can comment the following line in the code:

```
EXEC CICS START TRANSACTION("PMON") AFTER MINUTES(30);
```

The output of this program is put into a TDQ called *PMON*, as shown in Example 5-19.

Example 5-19 Output of the sample program monitor.ccs

```
=========================TIMESTAMP: 2005/11/1 -
17:26:05============================
Region Name: TESTRG System ID: ISCO
Storage Pool Information
RG Pool: Max size: 2097152 Thresh: 1887436 Cur Alloc: 102768
TS Pool: Max size: 1048576 Thresh: 943718 Cur Alloc: 46174

TaskDetail: TaskSeq| TASKID | PROCESSID | RUNSTATUS |TRANSACTION|
PROGRAM |STARTCODE
1     | 955  | 1036480   | RUNNING        | ITSK| INQTSK | TO
```

5.5.4 Monitoring TXSeries using IBM Tivoli Monitoring

We can monitor TXSeries components using IBM Tivoli Monitoring. Tivoli Monitoring enables you to monitor the availability and performance of the enterprise system in a single window. Through Tivoli Monitoring, you can customize the views for improved visualization of the enterprise system status. It provides different services, such as a notification mechanism, a user interface (UI) presentation, and communication services for raising the alerts.

Complete the following steps to monitor TXSeries using Tivoli Monitoring:

1. Check the prerequisites for Tivoli Monitoring.
2. Install IBM TXSeries Monitoring Agent on all machines where the TXSeries instance is running.
3. Install the IBM Tivoli Enterprise Portal Server and IBM Tivoli Enterprise Monitoring Server components of IBM TXSeries Monitoring Agent.
4. Configure TXSeries Monitoring Agent.
5. Set up the region for monitoring.
6. Start TXSeries Monitoring Agent.
7. Start the region.
8. Access TXSeries views from IBM Tivoli Enterprise Portal Server.

For more details about Monitoring TXSeries using IBM Tivoli Monitoring, see the following IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSAL2T_8.1.0/com.ibm.cics.tx.doc/tasks/t_monitoring_tm.html

Configuring the txdemo region for Tivoli Monitoring

To configure the txdemo region, perform the following steps:

1. Update the MD stanza of the txdemo region:

   ```
   cicsupdate -c md -r txdemo EnableTivoliMonitoring=yes
   TivoliMonitoringPort=<port number for ITM agent and region
   communication>  TivoliRefreshRate=<30 second is the minimum value
   and maximum value is 86400 (24 hours)>
   ```

2. You can configure IBM Tivoli Monitoring using the **itmcmd** command, where <*t8*> is the agent code for TXSeries Monitoring Agent:

   ```
   <path to tivoli home>/bin/itmcmd config -A <t8>
   ```

3. Stop and start IBM Tivoli Monitoring agent.
4. Stop and perform a cold start of the txdemo region.

Monitoring SFS using IBM Tivoli Monitoring

The SFS view of Tivoli Monitoring displays log volume and data volume page details. It also gives a graphical representation of data volume usage. You can create Tivoli Monitoring situations to raise alerts on the increased usage of log volume or data volume.

Large use of log volume indicates that there might be transactions in SFS with a `commitcomplete` state. If there are transactions in a `commitcomplete` state, and they do not seem to be changing state over time, it can be an indication of a problem.

You can use the TXSeries-supplied *watchtran* tool or **`tkadmin`** commands to monitor the transaction status in SFS. Increased usage of data volume indicates that transactions are writing a large amount of data into the SFS. To increase the data volume space, you can extend the data volume size.

Monitoring WLM using IBM Tivoli Monitoring

This workspace provides statistics information about TXSeries Workload Manager (WLM) component, and is available only on the AIX platform. The view gets populated only on the client-owning region (COR) node. It displays WLM routing details, and the status of regions that are part of the WLM configuration.

Selection and normal fields indicates the performance of AOR. This information can be used to visualize the distribution of work across different AOR regions. An increased number of abend counts at a particular AOR warns about configuration or health issues of the AOR.

Monitoring region logs using IBM Tivoli Monitoring

The ConsoleLog lists the latest `console.nnnnnn` file. The ConsoleLog is the primary log file for the activities that take place in the region. Although all of the messages from the user applications and libraries to `stderr` are redirected to this file, this view lists only the TXSeries runtime messages, avoiding the clutter.

The CSMTLog lists the latest `CSMT.out` file. `CSMT.out` is a TDQ that contains CICS messages about transactions or system errors, mostly related to terminals. Applications can also write to `CSMT.out` using **`EXEC`** CICS commands.

The CCINLog provides the listing of the latest `CCIN.out` file. The `CCIN.out` file is an external TDQ containing information about client connections. This file provides information about investigating issues related to connections getting dropped from the remote region or CTG connection.

The messages are color-coded, based on the severity of the message. You can sort the listing based on the various attributes of the message, such as `time stamp`, `process ID`, and so on.

Figure 5-26 shows the Console Log.

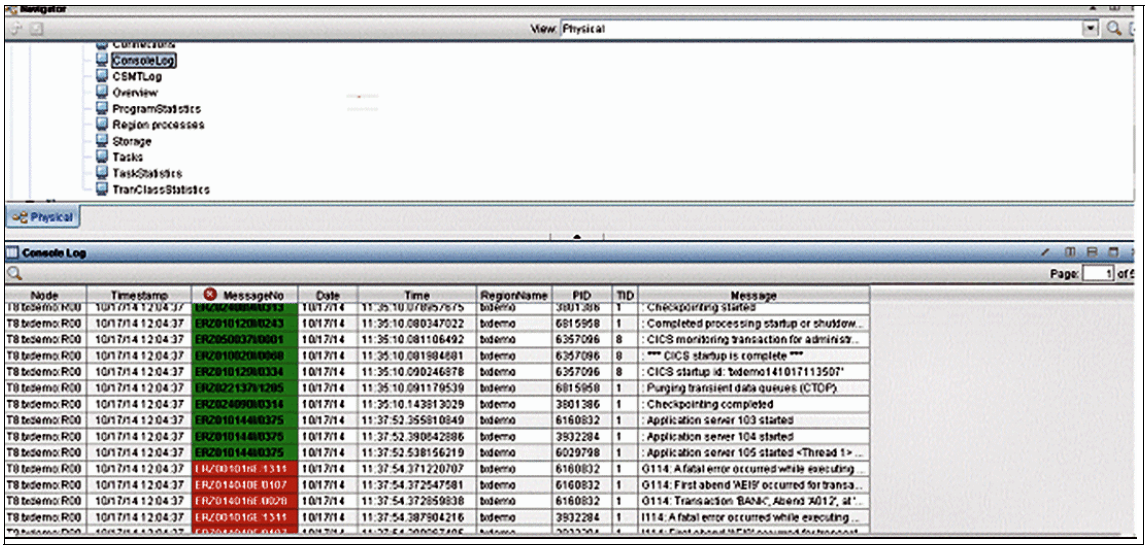


Figure 5-26 Console Log workspace

Monitoring region health using IBM Tivoli Monitoring

The Overview workspace of the region, shown in Figure 5-27, provides the basic region information, such as `region status`, `Max/Minserver`, `MaxRegion pool size`, and `XPRecvTimeout`. It displays the health of the region regarding abend counts, short on storage count for region pool, short on storage count for task shared pool, and so on. You can create situations to raise alerts for the different health parameters of the region.

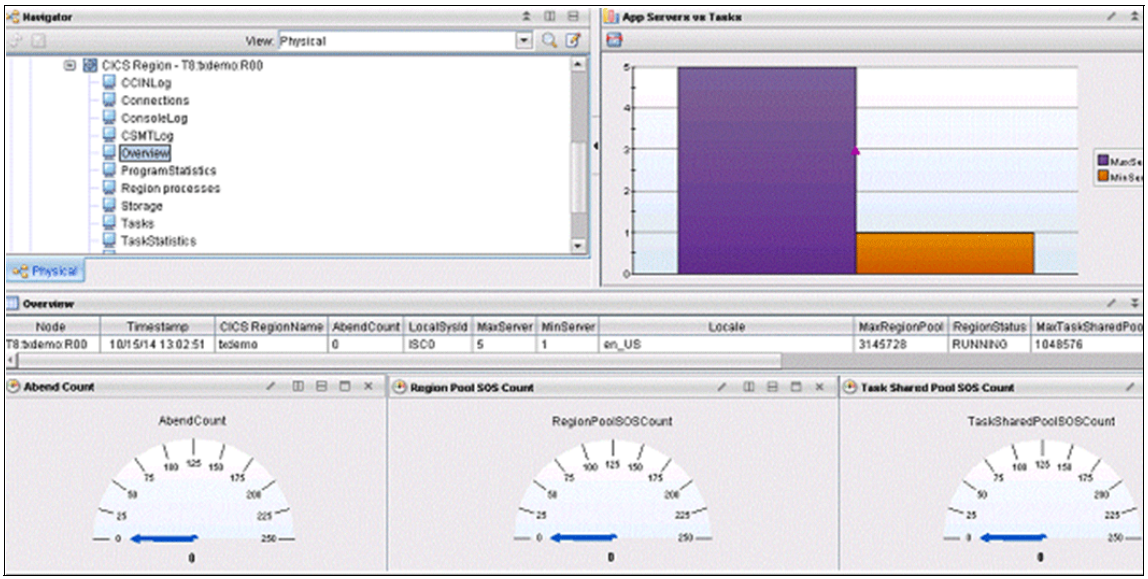


Figure 5-27 Region Overviews

Monitoring TXSeries programs using IBM Tivoli Monitoring

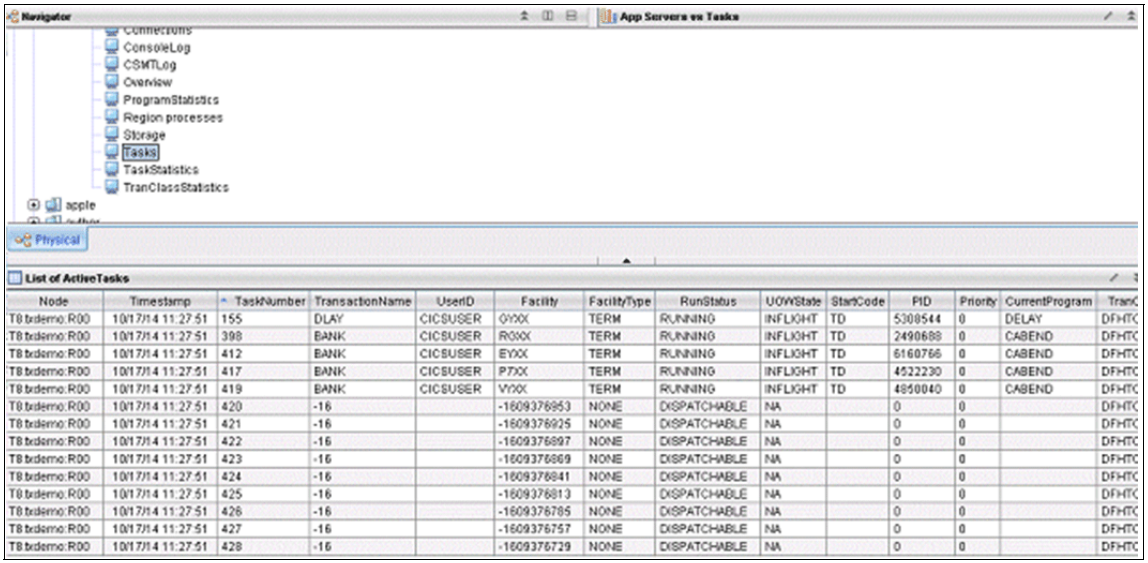
The ProgramsStatistics workspace lists the programs installed for each region. Runcount is displayed for each of the programs. This enables you to sort the programs in order of the number of invocations of the programs. This information can be used for performance tuning in terms of program caching. The most-used programs can be cached for better performance.

Monitoring tasks using IBM Tivoli Monitoring

The Tasks workspace lists the active tasks of the region. You can sort the tasks according to the task number, and identify the long-running tasks. The long-running tasks have smaller task numbers. The long-running transactions list indicates the probable hung transactions.

The `UOWstate` indicates the unit of work (UOW) state of the task. You can sort the tasks according to the `UOWstate`, and monitor the need to change the application server numbers (by changing the `min/maxserver` numbers). `TaskStatistics` provides the statistics of tasks run for the region.

TranClass Statistics provides the tranclass-related statistical information of the processes. It provides a graphical representation of tranclasses corresponding to the number of active tasks (Figure 5-28), which helps to identify the tranclasses for which the peak threshold limit has been reached. It helps to tune the tranclass definitions and the classification of tasks.



Node	Timestamp	TaskNumber	TransactionName	UserID	Facility	Facility/Type	RunStatus	UOWState	StartCode	PID	Priority	CurrentProgram	TranC
T8 txdemo:R00	10/17/14 11:27:51	155	DLAY	CICSUSER	GYXX	TERM	RUNNING	INFLIGHT	TD	5300544	0	DELAY	DFHTC
T8 txdemo:R00	10/17/14 11:27:51	398	BANK	CICSUSER	RGXX	TERM	RUNNING	INFLIGHT	TD	2490688	0	CABEND	DFHTC
T8 txdemo:R00	10/17/14 11:27:51	412	BANK	CICSUSER	EYXX	TERM	RUNNING	INFLIGHT	TD	6160766	0	CABEND	DFHTC
T8 txdemo:R00	10/17/14 11:27:51	417	BANK	CICSUSER	PTXX	TERM	RUNNING	INFLIGHT	TD	4522230	0	CABEND	DFHTC
T8 txdemo:R00	10/17/14 11:27:51	419	BANK	CICSUSER	VXXX	TERM	RUNNING	INFLIGHT	TD	4850040	0	CABEND	DFHTC
T8 txdemo:R00	10/17/14 11:27:51	420	-16		-1609376953	NONE	DISPATCHABLE	NA		0	0		DFHTC
T8 txdemo:R00	10/17/14 11:27:51	421	-16		-1609376925	NONE	DISPATCHABLE	NA		0	0		DFHTC
T8 txdemo:R00	10/17/14 11:27:51	422	-16		-1609376897	NONE	DISPATCHABLE	NA		0	0		DFHTC
T8 txdemo:R00	10/17/14 11:27:51	423	-16		-1609376869	NONE	DISPATCHABLE	NA		0	0		DFHTC
T8 txdemo:R00	10/17/14 11:27:51	424	-16		-1609376841	NONE	DISPATCHABLE	NA		0	0		DFHTC
T8 txdemo:R00	10/17/14 11:27:51	425	-16		-1609376813	NONE	DISPATCHABLE	NA		0	0		DFHTC
T8 txdemo:R00	10/17/14 11:27:51	426	-16		-1609376785	NONE	DISPATCHABLE	NA		0	0		DFHTC
T8 txdemo:R00	10/17/14 11:27:51	427	-16		-1609376757	NONE	DISPATCHABLE	NA		0	0		DFHTC
T8 txdemo:R00	10/17/14 11:27:51	428	-16		-1609376729	NONE	DISPATCHABLE	NA		0	0		DFHTC

Figure 5-28 Task statistics view

Monitoring storage using IBM Tivoli Monitoring

The Storage workspace shown in Figure 5-29 indicates the storage status of the region. It displays the RegionPool Usage and TaskShared pool Usage. Storage attributes of the region can be customized according to the storage view. This information can be used to tune the storage parameters of a region.

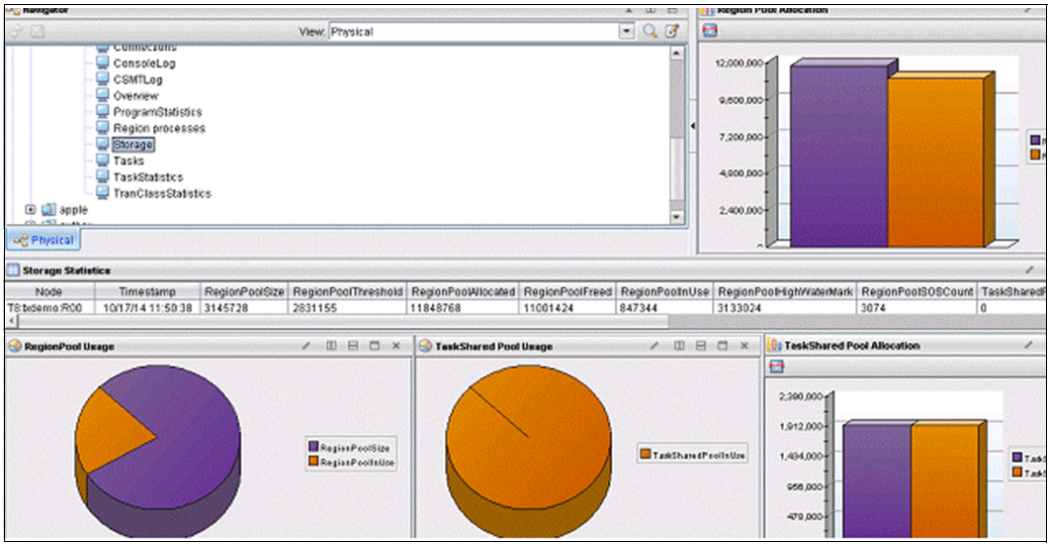


Figure 5-29 Storage view

Setting alerts in IBM Tivoli Monitoring

Tivoli Monitoring enables you to invoke an action based on the value of a monitored field, or a combination of fields. This can be defined using a Tivoli Monitoring situation, which is a definition of a set of conditions. When the condition becomes true, an action is taken to raise the alert. By default, this alert is usually indicated in the UI (Tivoli Enterprise Portal) by colored notification icons. Tivoli Monitoring also supports sending alerts using email or SMS alerts.

Follow these steps to define a Tivoli Monitoring situation to alert on abendcount:

1. Right-click the view name **Overview** and click **Situations**.
2. Click the **New Situation** icon at the top of the menu.
3. Enter the name and description of the situation and click **OK**.
4. Select the **AbendCount** attribute and click **OK**.
5. For the formula definition, enter the condition as greater than 5.
6. Enter the sampling interval (for example, 1 minute).
7. Click **OK**.
8. Notice that a red alert is displayed when the abend count is greater than 5.

For more details, see the Tivoli Monitoring IBM Knowledge Center.

5.5.5 Transaction tracking

If your transactions are spanned only in the region, you can use TXSeries-supported monitoring facilities for tracking the transaction execution-specific information, such as start time, stop time, end time, and so on. This helps you to identify the hanging transactions. However, if your transactions are spanned across multiple regions, the monitoring facilities alone do not help to debug the transactions.

They don't help because transaction name, program name, task ID, and so on, change in each region during the transaction execution. Using the transaction tracking facility, transaction execution in each region can be tracked, and can also be mapped to identify the transaction origin.

Using the transaction tracking facility, you can know the peer transaction name, peer region name, peer user ID, peer IP address, process ID of the peer CICSAS process, client/peer IP address, and client port. The transaction tracking facility is useful in debugging the following scenarios:

- Tracking time spent on each region if the transaction spans across multiple regions
- Transactions hanging in a region due to waiting for a response from a partner region
- Forcepurge as a result of client timeout
- Identifying the terminal ID from where the transaction got initiated
- Identifying branches or end clients that are initiating the transactions
- Identifying the workload pattern from partner and peer regions

To enable transaction tracking between two regions, you need to enable the following attributes for the connection for which you want the information:

- `EnableTaskInfo` attribute to `yes` in RD stanza in the front-end region
- `SendTaskInfo` attribute to `yes` in CD stanza in the front-end region

Restriction: The transaction tracking facility is *only* supported with `CICS_TCP` and `PPC_TCP` protocols.

6

Troubleshooting

In this chapter, we describe what to do when things do not happen as expected. It is important to remember that problems can be with related or connected software packages or with IBM TXSeries itself.

We consider the difficulties involved in knowing where the core problem is, and describe a basic methodology for finding it. For cases where the problems lie within the realm of TXSeries, we have supplied a description of the tools that you can use to investigate.

This chapter provides information about the following topics:

6.1 Common methodology

There are different approaches that you can take to diagnose problems. The problems are often challenging in a distributed middleware environment. In this section, we describe the common methodology that you can use to diagnose a problem.

6.1.1 Determining the source of the problem

When client, middleware, and back-end components work together, it is easy to follow the wrong diagnostic path. It is important to remember that the problem can be anywhere, not just in TXSeries. The following sections list some of the steps that you can consider.

Understanding the entire system setup

To understand the system setup, perform the following tasks:

- Inventory all machines involved and, for each, the associated operating system (OS) and product versions.
- Inventory all communication components, such as protocols, port numbers, and so on.
- Inventory all software packages and their versions.

Tracking back to the source of the error

Consider the scenario where an error message is reported by some part of the integrated system. Work backwards from the error message through the various layers to determine the component that is the source of the issue. It is important to correlate different sets of logs generated by the products involved, to isolate the cause of the problem. In most of the scenarios, you can do this by *matching the time stamp in different sets of logs*.

6.1.2 Solving the problem

The exact steps required to solve the problem are different depending on which component turns out to be at the root of the problem. If TXSeries is at the root of the problem, see the following section.

6.2 TXSeries problem determination

Problem determinations on TXSeries systems usually start with a symptom, or set of symptoms, and you trace them back to their cause. In this section, we describe tools and techniques that you can use to find the cause of a problem, and suggest actions for solving the problem.

6.2.1 Introduction to problem determination

Working through a problem includes these three steps:

1. Look for information to help diagnose the cause, as described in 6.2.2, “Sources of information” on page 332.
2. Understand the cause of the problem.
3. If possible, fix the problem. Otherwise, report the problem.

Preliminary checks

If one of the following situations applies to you, consider a clear course of action, or consult the TXSeries documentation library for the correct course of action:

- Are you the first person to have a problem starting the IBM Customer Information Control System (IBM CICS) region?

  Check for the information about CICS published on the TXSeries support site, the TXSeries documentation library for your product version, IBM DWAnswers, and blogs. You can use IBM Software Assistant or any public Internet search engines to search the errors.

  Include “IBM TXSeries” along with the error symptoms for better search results. The product’s social media pages are updated regularly with many configuration issues and tips.

- Are there any messages explaining the error?

  If there is an error message associated with the symptom, use the **`cicserr`** command for more information.

- Can you reproduce the error?

  If the error is reproducible, the next action is determining if it is due to a configuration issue, an application issue, or a product issue.

  See the *Troubleshooting* task in the TXSeries documentation library for more details.

- Does the error occur at specific times of the day?
- Is the error intermittent?
- Have any changes been made since the last successful run?

- Are specific parts of the network affected by the problem?
- Has anyone run the application successfully before?
- Does the problem involve software other than CICS?

Classifying the problem

The following list shows the possible classifications of problems in this chapter:

- Problems caused by CICS system or transaction abnormal terminations
- Problems caused by a wait, a loop, or by poor performance
- Problems caused by waits
- Problems caused by loops
- Performance issues
- Unanticipated output
- Problems caused by storage violations
- Problems involving Encina or the operating system
- Problems involving databases

  These are described in the TXSeries documentation library:

  http://www.ibm.com/support/knowledgecenter/SSAL2T/welcome

> **Information:** To learn about CICS user exits and problem determination, see Chapter 5, “System customization” on page 247.

6.2.2 Sources of information

This section describes where to obtain information about the cause of a problem.

Directories and files containing CICS diagnostic information

As you work through the problems, knowing about certain key TXSeries directories and files makes your work easier:

- Region directory. Every TXSeries region you define has a central location where special files and subdirectories are found:
  - Open systems: `/var/cics_regions/<region_name>`
  - Microsoft Windows: `<install_drive>\var\cics_regions\<region_name>`
- Files found in the region directory:
  - `console.nnnnnn`. This is the primary log file for activities taking place in the region. All messages to `stderr` are redirected to this file, for example, error messages from Common Business Oriented Language (COBOL) programs. The size of the file is limited by the `MaxConsoleSize` in the region definition (RD) stanza. The default value for `MaxConsoleSize` is `0` (unlimited).

> **Remember:** A new console.*nnnnnn* file (where *nnnnnn* is a sequential number) is created every time the region is started. To identify the current console.*nnnnnn* file, review the contents of the console.nam file.

> **Tip:** Use the **cicstail -r RegionName** command from the OS prompt to observe messages that are added to both the console.*nnnnnn* and CSMT.out files as your application runs.

  - symrecs.*nnnnnn* (symptom records). This file logs the messages that are related to potential software or system damage. Messages in this file can assist your IBM support organization. This file does not exist until a record is written to it.
  - console.msg. All messages to stdout are redirected to this file. For example, a COBOL program that uses **DISPLAY MY-ITEM UPON CONSOLE**.
  - <*region name*>.env. Logs the environment under which the CICS subsystem started.
- Region data directory:
  - Open systems: /var/cics_regions/<*region_name*>/data
  - Windows: <*install_drive*>\var\cics_regions\<*region_name*>\data
- Files in the region data directory:
  - CSMT.out. This is a transient data queue that contains CICS messages about transactions or system errors. Applications can also write to CSMT.out using **EXEC CICS** commands.
  - CCIN.out. This is a transient data queue that contains messages about client connections.
  - statsfile. This file contains CICS region statistics output. You can format the files using the cicssfmt utility.
- Region dumps/dir1 directory:

  The region's dumps/dir1 directory is the current working directory for all of the CICS processes. Therefore, by default, you can find any files generated or created by third-party software running under CICS in this location:
  - Open systems: /var/cics_regions/<*region_name*>/dumps/dir1
  - Windows: <*install_drive*>\var\cics_regions\<*region_name*>\dumps\dir1

- Files in the region `dumps/dir1` directory:
  - `cics<pid>.traceback`. This file logs function stack information when an incorrect exception (`SigILL`, `SigSEGV`) occurs in an application or CICS internal code.
  - `SYSA<nnnn>.dmp<mm>`. This is a system dump file that contains region-specific information generated on the abnormal termination of the region.
  - `ASRA<nnnn>.dmp<mm>`. This is a transaction dump file that contains transaction-related information generated when an abnormal end of task (abend) occurs in the transaction.
  - `core<pid>.<time stamp>`. This file is generated by the OS when a CICS process ends abnormally. CICS then attempts to format the generated core file. If successful, this creates a file with the same name as the core file and suffix of `.fmt`. The formatted file contains the process/thread stack.

6.2.3 Dealing with abnormal terminations

In the case of an abnormal CICS transaction termination, consider the following tools:

- See 6.2.2, “Sources of information” on page 332.
- CICS traces and dumps, as discussed in 6.2.13, “Using traces in problem determination” on page 343.
- Language debugging tools, as described in the following sections.

6.2.4 Distinguishing between loops, poor performance, and waits

Loops, poor performance, and waits can have similar symptoms:

- One or more tasks in CICS are in one of the following states:
  - Cannot start
  - Remain suspended
  - Cannot complete
- The output is missing.
- Terminal activity is reduced or stopped.

Looping

To determine whether the problem is a program that is looping, look for these additional clues:

- Processor usage is very high.
- Repetitive output on terminals, in temporary storage queues, or in data files and CICS journals.
- A task demands excessive storage.
- Statistics show a large number of file accesses for an individual task.

Poor performance

A performance issue can be because of capacity problems in CICS. If you receive messages saying that CICS is under stress, this might be the case. However, verify that you do not have problems associated with loops before concluding that it is actually a capacity issue.

Waits

A wait condition does not have the conditions associated with a loop. However, you do not receive messages regarding CICS under stress as you might with a performance issue.

Restriction: These are typical guidelines, and not always a perfect map to the correct conclusion.

6.2.5 Dealing with waits

There are various kinds of waits. For a complete description of each type, see the TXSeries documentation library. In this section, we describe each type briefly:

- CICS system waits. These can be caused by legitimate activities:
  - Deadlocks. Check your preference for the `DeadLockTimeout` setting in the transaction definitions (TD).
  - Back-end transaction not starting. Configure the communications definitions (CD) as preferred for possible improvements.
  - OS causes, such as waits for a file on an overloaded network.
- Maximum server condition waits. If there are tasks waiting for an application server, and the number of applications servers has reached the `MaxServer` limit specified in the RD, the task must wait. You can raise the limit, if required.

- Terminal waits. These can be due to, among other causes, the following human factors:
  - A terminal user not responding to a request for input.
  - A terminal is being debugged by another terminal that is using CICS execution diagnostic facility (EDF), and the user is entering a required response.
- Intersystem waits. These can be either a Transmission Control Protocol/Internet Protocol (TCP/IP) or Systems Network Architecture (SNA) connection. Use CICS monitoring facility (CMF) for information gathering.
- Transient data waits. If you program queue operations without the `NOSUSPEND` option, such operations can cause one task to wait for another. Use CMF to determine whether a task is waiting on transient data input or output.
- File control waits. Depending on your choice of programming method, file control waits can be longer or shorter. Check whether a task is waiting for file control by using CMF.
- Temporary storage queue waits. There are two reasons for such waits:
  - Insufficient auxiliary storage or main storage to satisfy a temporary storage queue request.
  - Another transaction is already using the temporary storage queue.

  You can check whether specific tasks are frequently waiting on temporary storage queues by using CMF. You can also ensure that queue identifiers are unique, so that different tasks do not unintentionally perform operations on the same queue.
- System dump waits. Waits during a system dump are normal.
- `ENQ` and `SUSPEND` task control waits. For details about this wait, see the TXSeries documentation library, and consider how application programming design changes can help.
- Journal waits. Verify whether tasks are frequently waiting on journals by using CMF.
- Syncpoint waits. Syncpoint waits occur when tasks issue **`EXEC CICS SYNCPOINT`** requests in synclevel 2 conversations. Tasks wait until a sync point has either committed or rolled back any recoverable resources involved.
- CICS system process waits. See the TXSeries documentation library.
- CICS has stalled. See the TXSeries documentation library. If CICS has stalled during a run, and not during initialization or termination, the problem determination guide offers suggestions about RD parameters that might require your attention.

6.2.6 Dealing with loops

This section lists some of the ways to identify a loop:

- Loops in a CICS environment perform either of the following actions:
  - Return control to CICS by making a call to a CICS application programming interface (API). In this case, tracing can help define the limits of the loop.
  - Do not return control to CICS, making no calls to the CICS API. In this case, you can add **`EXEC CICS ENTER`** to the sections of the programs that you suspect, and then run a trace.
- To help identify a task that might be looping, you can set the `MaxTaskCPU` and the `MaxTaskAction` attributes in the RD to give a warning for (or even end) a looping task that happens to be central processing unit (CPU)-intensive. Use the `MaxTaskCPU` attribute of the TD to target specific transactions.
- Alternatively, and possibly primarily, you can use the interactive debugger associated with your programming language to pinpoint the cause of the loop. See “6.3, “Debugging application programs” on page 350.

After you identify the cause of the loop, determine the appropriate course of action.

6.2.7 Dealing with performance problems

For details about how to deal with performance problems, see “Dealing with performance problems” in the *TXSeries documentation library*. The following types are the basic performance problem categories:

- Scheduling bottlenecks. There are two categories of scheduling bottlenecks:
  - Tasks not yet given to the transaction scheduler. This might have to do with either of the following items:
    - Terminal and remotely initiated tasks
    - Interval control transactions
  - Tasks not scheduled by the scheduler:
    - The task is held by a class.
    - No application server is available.
    - The task has priority issues.
- Short on storage (SOS).

- Terminal definitions not removed from the system.

  If users routinely end their cicsterm, the associated, autoinstalled TERMINAL might not be removed. As these accumulate, performance can suffer. The chapter about dealing with performance problems in the TXSeries documentation library provides a method of stopping the leftover processes.

- Incorrect settings of region attributes:
  - `MinServers`
  - `MaxServers`
  - `CheckpointInterval`
  - `IntrospectInterval`

- Incorrect settings of Structured File Server (SFS) attributes:
  - `OpThreadPoolSize`
  - `ResThreadPoolSize`
  - `IdleTimeout`

6.2.8 Dealing with unanticipated output

Unexpected output falls into three categories:

- An output device displays unanticipated data.
- Unanticipated data is present on a file or user journal.
- An application did not work as expected.

> **Tip:** If you face one of these situations, you can find useful information in the chapter about dealing with unanticipated output in the TXSeries documentation library. The following sections provide an outline of the chapter to introduce you to the topic.

An output device displays unanticipated data

- Preliminary information
- Specific types of unanticipated data:
  - Unexpected messages and codes.
  - Unexpected appearance of uppercase or lowercase characters.
  - Incorrect data values are displayed.
  - Some data is not displayed.
  - The data format is wrong.

Unanticipated data is present on a file or user journal

The unanticipated data is similar to the types listed previously.

An application did not work as expected

When an application did not work as expected, the following approaches can be useful:

- General points to consider
- Using traces and dumps
- Classifying the problem
- No output at all:
  - Are there any messages?
  - Can you use the terminal where the transaction should have started?
  - What to do if the task is still in the system.
  - What to do if the task is not in the system.
  - Did the task run? Techniques for all tasks:
    - Using CICS system trace entry points
    - Using the EDF
    - Using statistics
    - Using the CICS Execute Temporary Storage Browse (**CEBR**) command to view the contents of the temporary storage queue (TSQ)
    - Using the CICS Execute Command-Level Interpreter (**CECI**) command
    - Disabling the transaction
  - Investigating tasks initiated by automatic transaction initiation (ATI).
- Incorrect output:
  - Missing or incorrect records
  - Data mapped incorrectly into the program
  - Poor programming logic
  - Data mapped incorrectly to the terminal

6.2.9 Dealing with storage violations

Storage violations consist of two categories:

- Violations detected and reported by CICS

  Violations detected by CICS are easy to identify, because CICS sends an informative message to the `console.`*`nnnnnn`* file or to the CICS System Termination (**`CSMT`**) TD queue. The chapter about dealing with storage violations in the TXSeries documentation library contains an extensive procedure to determine the source of the problem, including the use of dumps and traces.

  When CICS indicates that a storage violation is clearly associated with a specific transaction, you might prefer to use the debugging tool specific to your programming language to find the cause.

- Violations not detected by CICS

  It is not easy to identify storage violations that are undetected by CICS. They can cause almost any sort of symptom. The chapter about dealing with storage violations in the TXSeries documentation library has a section about storage violations that affect innocent transactions. This section describes how to use tracing to find the problem.

6.2.10 Dealing with memory and file descriptor leaks

You might find that by using operating system tools or CICS messages, the CICS application server memory continuously grows over a period of time. The CICS `AppProbe` facility turns on a monitoring facility that you can use to analyze this memory growth. It provides the ability to see memory and file descriptor information for specific transactions at the following points of operation:

- User application load and unload
- User application entry and exit
- **`EXEC CICS`** API entry and exit

With this capability, it is possible to determine whether a leak is in your own application code or in a CICS operation. If it is within your own code, you can apply various methods to fix the problem, including the use of the debugger for your programming language. If the leak occurs within CICS, contact your IBM TXSeries support organization.

For complete information about how to use this facility, see the chapter about dealing with memory and file descriptor leaks in the TXSeries documentation library.

6.2.11 Dealing with problems involving the operating system

TXSeries operates atop the facilities provided by the operating system.

Problems involving databases are generally caused by one of the following errors:

- Errors in CICS configuration
- Errors in the relational database management system (RDBMS) configuration
- Application coding errors
- Application building errors

Before you start investigating, ensure that you understand how CICS uses relational databases. See the topic about configuring in the TXSeries documentation library for details.

Checking CICS and RDBMS configuration

You can perform the following tasks to check for problems involving databases:

- Check the database error logs and CICS message destinations (`console.nnnnnn`, `CSMT.out`, and `symrecs`).
- Check that you have correctly run the steps provided in the Configuring task in the TXSeries documentation library.
- Use **`pview`** (Windows) or **`ps`** (OpenSystems) to check the status of processes.
- Look for any release-specific information in the product README file, supplied in `<TX_install_directory>/cics/doc`. This file might contain changes to the database enabling procedure.
- Check that you have installed the correct level of software. To verify, see the Supported Software Environment on the support page of the TXSeries for Multiplatforms IBM Knowledge Center.
- Run the provided CICS database sample programs to verify that your database has been set up correctly. See the *CICS Administration Guide* for details.
- Check for Structured Query Language (SQL) codes handled by the application, as shown in Example 6-1, for example, for IBM DB2.

Example 6-1 Checking the SQL codes

```
EXEC SQL WHENEVER SQLERROR GOTO :ERR-EXIT END-EXEC.
ERR-EXIT.
*****************
EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
MOVE SQLERRMC OF SQLCA TO MESSAGEO OF PANEL4O.
MOVE SQLCODE OF SQLCA TO CODEO OF PANEL4O.
```

```
EXEC CICS SEND
  MAP ('PANEL4')
  MAPSET ('UXA1')
  FREEKB
  ERASE
END-EXEC
```

If you are using DB2 for file control, the following information might be useful:

– The `cicsddt` utility for loading, unloading, and listing DB2 files. It shows the database files in the context of the CICS organization. See the configuring task in the TXSeries documentation library for details.

– Many of the errors generated by the `DamDB` code are also written out to CSMT. Where SQL error codes are returned, these are converted to string explanations and inserted into the messages.

Checking application coding

All relational databases impose restrictions on SQL allowed in an extended architecture (XA) environment, for example, no transaction control commands, such as **`SQL COMMIT`**. The problem often manifests itself only at run time. Check your database documentation for the restrictions for your version of the database. See the programming task in the TXSeries documentation library for information about coding for databases.

Checking application building

You must consistently build the switch-load file, applications, and COBOL runtimes. For example, this means that you must link the same libraries consistently in all the modules that you built and are run under TXSeries environment. Use **`dump -h`** to check consistency. Use the provided CICS sample programs, as a template. These samples illustrate a consistent build.

Additional information

The chapter about dealing with database problems in the TXSeries documentation library has this additional section, to which you can refer:

- DB2
- Informix
- Oracle
- Sybase

6.2.12 Resolving problems with CICS clients

See the Troubleshooting topic in the TXSeries documentation library for further information about how to resolve problems with CICS clients.

6.2.13 Using traces in problem determination

Tracing is a way of collecting detailed information about how a process runs. It consists of reporting information about events in a process. Events include the calling of and return from functions, the exit of processes, and other significant occurrences in the execution of a program.

You can use trace information to debug an application, and to tune the operation of a product. Tracing can assist both application developers and system administrators, and it is often crucial when asking for product support.

CICS trace types

CICS provides two types of tracing:

- Application trace. This consists of two categories:
  - EXEC trace. Traces all **`EXEC CICS`** commands with their options.
  - User trace. Enables application programmers to debug and investigate their programs. User trace is the trace of an individual task. It is written while a task is running. The user trace events are triggered through the **`EXEC CICS ENTER`** command.
- System trace: This consists of product trace which tracks significant events in CICS, itself. There are five trace levels.

Enabling and requesting CICS trace

The tracing done in a CICS region is governed by a combination of the RD attributes that enables tracing and explicit requests for tracing. The RD attributes that enable tracing do not cause the collection of trace data. They just permit or forbid it. The tracing requests collect trace data, but only when trace collection is enabled. If trace collection is disabled, no tracing requests are performed, and if tracing is enabled but no requests are made, no trace is collected.

Application trace: Enabling and requesting

To enable tracing in the region, complete the following steps:

1. Turning on the `TraceFlagMaster` RD attribute.

   You can do this administratively, from within a CICS transaction program by using the **`EXEC CICS TRACE ON`** command. Alternatively, you can do this from a terminal with the **`CEMT SET TRACE ON`** command.

   > **Important:** `TraceFlagMaster` is hierarchically superior to all of the other trace-related RD entries. When it is *on*, the others apply. When it is *off*, the others have no effect.

2. Enable, as required, user and exec level application tracing:
   - To enable user trace, turn on the `TraceFlagUser` RD attribute. You can do this administratively, or from within a CICS transaction program by using the **`EXEC CICS TRACE ON USER`** command.
   - To enable exec trace, turn on the `TraceFlagExec` RD attribute. You can do this administratively, or from within a CICS transaction program by using the **`EXEC CICS TRACE ON EI`** command.
3. Request collection of application trace events by using the **`EXEC CICS TRACE ON USER SINGLE`** command within the transaction program. The command turns on the collection of user or exec trace, depending on the settings of the application trace flags. The request is cancelled automatically when the transaction program exits.

   Application trace events are triggered when the CICS API command EXEC CICS ENTER is started. The request to collect application trace can be made only from within a program, there is no administrative equivalent.

System trace: Enabling and requesting

To enable tracing in the region, complete the following steps:

1. Turn on the `TraceFlagMaster` RD attribute. You can do this administratively, from within a CICS transaction program by using the **`EXEC CICS TRACE ON`** command. Alternatively, you can do this from a terminal with the **`CEMT SET TRACE ON`** command.
2. Enable system trace for the region by turning on the `TraceFlagSystem` RD attribute.
3. Request collection of the system trace by setting the `TraceSystemSpec` RD attribute to indicate the events, modules, and processes that you want to trace. The basic form of the TraceSystemSpec is:

   ```
   TraceSystemSpec="TraceDefn[,TraceDefn]"
   ```

Introduction to TraceDefn

Each `TraceDefn` consists of a left side and a right side separated by an equals sign (=). There are two valid types of `TraceDefn` trace definitions:

- Those that list trace levels for CICS modules, which take the following form:

  `moduleList=traceLevel`

- Those that list CICS process types to trace, which take the following form:

  `proc=processTypeList`

The following steps illustrate a common way to manage system trace:

1. Set the trace specification for all modules to level `0`.
2. Turn off all tracing.
3. Selectively trace specific modules at appropriate levels.

For example, consider the following `TraceSystemSpec`:

```
TraceSystemSpec=
"all=0,damtd+damfi=4,proc=as,taslu=4,proc=ip+rl,suppr+comsu=4"
```

TraceSystemSpec has the following features:

- Turns off system tracing for all modules
- Turns on tracing, at trace level `4`, in the `damtd` and `damfi` modules in all of the processes in the region.
- Turns on tracing for the `taslu` module at level `4` within the application server processes (`proc=as`).
- Turns on tracing for the `suppr` and `comsu` modules at level `4` within the IP and Remote Procedure Call (RPC) listener processes (`proc=ip+rl`).

Information: For a complete explanation of the format of `TraceSystemSpec`, see the chapter about using CICS trace in the Troubleshooting task of the TXSeries documentation library.

Application trace: File name and location

When application trace is collected, the trace records are written to files. The trace output directory is determined by the `TraceDirectoryUser` entry in the RD stanza. The following defaults exist, depending on the OS:

- UNIX: `/tmp`
- Windows: `<`*`installDrive`*`>:\var\cics_tmp`

The output files are named according to the following pattern: `<`*`RegionName`*`>.<`*`UserName`*`>.n<`*`processNumber`*`>.p<`*`processID`*`>.cicsusr` (for example, `CICSREG.CICSUSER.n101.p204576.cicsusr`).

For programs that use user trace but run without a user identifier (such as programs that run at startup) the trace output directory is determined by the `TraceUserPublicFile` entry in the RD stanza. The default is `cicspubl`. The output files are named according to the following pattern:

`<regionName>.public.cicsusr (for example, CICSREG.public.cicsusr)`

System trace: Output location

System trace records are sent to one of several destinations set in RD stanza:

- `TraceFlagBuffer`. Sends trace records to the trace buffer. The default is `on`.
- `TraceFlagAux`. Sends trace records to the auxiliary files. The default is `off`.

Information: Extensive information about setting up system tracing is available in the TXSeries documentation library. In practice, a support technician from IBM is likely to determine the settings.

Formatting CICS traces

To format the CICS traces, go to the location (or directory) where the trace file is located and issue the following command:

```
cicstfmt {cicstrace file name} > {cicstrace file output}
```

6.2.14 Using dumps in problem determination

CICS dumps provide a detailed snapshot of what happens in CICS at the moment that the dump occurred. For example, you can look for the following information:

- The number and list of tasks running in the region.
- The number and list of application servers that are idle.
- The number and list of tasks that are queued and not yet scheduled.

Types of CICS dumps: Transaction dump

A transaction dump writes specified areas of memory to a file, to help you debug an application program, or identify why an abnormal termination or storage violation occurred. Transaction dumps can be generated in the following ways:

- **`EXEC CICS DUMP`**
- **`EXEC CICS ABEND`**
- Abnormal termination of transactions, such as ASRA or ASRB abends

Remember: Not all abends generate a dump.

Types of CICS dumps: System dump

A system dump writes information about the entire CICS region:

- Details of the last CICS command executed
- Details of each transaction in progress
- The region configuration at the time that the dump is taken
- Transaction dump of any non-application server process
- All enabled trace information

System dumps can be generated by the following events:

- CICS system shutdown
- Abnormal CICS system terminations
- **CEMT PERFORM SNAP**

> **Note:** A SNAP dump writes information about the entire CICS region, and can be initiated through a CICS terminal.

Types of CICS dumps: Core dump

In exceptional circumstances, CICS can generate an operating system core dump rather than a CICS formatted dump. Messages are written to the console, providing the reason for the core dump and the file name. CICS attempts to format the dump using `showProcInf`.

Dump file names have the following pattern: `<aaaa><nnnn>.dmp<mm>`.

The following items describe the dump file name components:

- `<aaaa>` is one of the following segments:
  - `ASRA` as a result of an `ASRA` abnormal termination
  - `ASRB` as a result of an `ASRB` abnormal termination
  - `SYSA` as a result of an `SYSA` abnormal termination
  - `SHUT` from a shutdown request
  - `SNAP` from a **CEMT PERFORM SNAP DUMP** request
  - Four-letter dump code from an **EXEC CICS DUMP** command
  - Four-letter abnormal termination code from an **EXEC CICS ABEND** command or abnormal transaction termination initiated by CICS
- `<nnnn>` is the dump sequence number. It is incremented with each dump that is performed.

> **Tip:** The current dump sequence number is saved at the time of region shutdown for the next time the region is started.

- `<mm>` numbers the files of a dump, which can require multiple files.

Setting the dump destination

Dump destination is controlled by the following settings:

- The `DumpName` attribute of the RD is the directory (containing the subdirectories) to which CICS dumps are written. The default is `/dumps`.
- The `CoreDumpName` attribute of the RD is the subdirectory of the `DumpName` directory. CICS uses this subdirectory for a core dump in the event of an unrecoverable CICS abnormal termination. The default is `/dir1`.

By default, CICS dumps are placed in the `/var/cics_regions/<regionName>/dumps/dir1` directory. You can enable dumps in the following ways:

- By setting the following attributes in RD and TD stanza files:
  - `SysDump` (RD) produces a dump on shutdown, `SNAP` and `ASRx` abends.
  - `ABDump` (RD) produces a system or transaction dump on `ASRB` abends.
  - `PCDump` (RD) produces a system or transaction dump on `ASRA` abends.
  - `TransDump` (TD) produces a transaction dump if a transaction abends.
- Using **`CEMT`** during run time:
  - **`CEMT INQUIRE/SET DUMP`**
  - **`CEMT INQUIRE/SET DUMPOPTIONS`**
- Through Dump Request User Exit (UE052017). The dump request user exit can be started at the following points:
  - When the **`EXEC CICS DUMP`** command is issued
  - When a transaction abend occurs
  - When a **`CEMT PERFORM SNAP`** is requested
  - When the region is shut down
  - When a system abend occurs

Controlling the dumps

Table 6-1 shows the steps that you can perform to control the dumps.

Table 6-1 Controlling the dumps

Step you want to take	Set the following attributes
Enable dump when the following events occur: ► CICS shuts down abnormally ► **CEMT PERFORM SNAP** is requested ► ASRA or ASRB abend occurs	SysDump = yes or **CEMT SET DUMP ON**
Produce a CICS system dump on an ASRA abend.	► SysDump = yes or **CEMT SET DUMP ON**, plus ► TransDump = yes, plus ► PCDump = yes or **CEMT SET DUMPOPTIONS PCDUMP**
Produce a CICS system dump on an ASRB abend.	► SysDump = yes or **CEMT SET DUMP ON**, plus ► TransDump = yes, plus ► ABDump = yes or **CEMT SET DUMPOPTIONS ABDUMP**
Produce a dump of CICS RD and main storage areas related to a task.	**EXEC CICS DUMP**
Produce a CICS transaction dump on an ASRB abend.	► TransDump = yes ► ABDump = no or **CEMT SET DUMPOPTIONS NOABDUMP**
Produce a CICS transaction dump on an ASRA abend.	► TransDump = yes ► PCDump = no or **CEMT SET DUMPOPTIONS NOPCDUMP**

Formatting the dumps

To format the dumps:

1. Navigate to the location (or directory) where the dump is generated.
2. Issue the following command:
   ```
   cicsdfmt {dump file name} > {dump file output}
   ```

6.2.15 Working with your support organization to solve your problem

For a full description of working with your support organization, see the TXSeries documentation library.

6.3 Debugging application programs

TXSeries provides tools to help you debug your application problems. In this section, we describe the various supplied transactions and tools that assist you in debugging your application programs.

6.3.1 CICS-supplied transactions for debugging CICS applications

Table 6-2 lists the CICS transactions for debugging CICS applications.

Table 6-2 CICS-supplied transactions for debugging

	IBM AIX	Windows	Linux on x86	Solaris	Hewlett-Packard UNIX (HP-UX)
C	CDCN	CDCN	X	X	X
C++	CDCN	CDCN	X	X	X
IBM PL/I	CDCN	CDCN	N/A	N/A	N/A
IBM COBOL	CDCN	CDCN	N/A	N/A	N/A
MF COBOL	CADB	CADB	CADB	CADB	CADB
Java	CJDB	CJDB	N/A	N/A	N/A

6.3.2 CICS execution diagnostic facility

The CICS transaction called CICS EDF gives the programmer the ability to observe each call that is made to the CICS API during a transaction. The CICS program stops before and after each **EXEC CICS API** call, and displays all parameters that are passed between CICS and the program.

Application program memory is available to be viewed, in character and hex dump formats only (no program symbols or field names are shown). No program code is presented. If that is required, use the language-specific debugging method.

To use the CICS EDF, perform the following steps:

1. Translate and compile the program with **cicstcl** and the **-e** flag.

   Important: Perform the translation of your CICS program with **-e** only if you intend to use CICS EDF, because it generates a performance cost.

2. Turn the EDF on by issuing the **CEDF <term>,ON** command.

   In this case, *<term>* is the terminal ID where CICS EDF is to be active. To make it default to the current terminal, issue the **CEDF,ON** command.

3. Begin the specific CICS transaction that you want to diagnose at the terminal specified. The window shown in Figure 6-1 opens.

```
TRANSACTION:   BANK PROGRAM:   MAINMENU TASK NUMBER:   0000000027 DISPLAY:   00
 STATUS:    PROGRAM INITIATION

     EIBTIME      = 154541
     EIBDATE      = 105304
     EIBTRNID     = 'BANK'
     EIBTASKN     = 27
     EIBTRMID     = 'DUEO'

     EIBCPOSN     = 4
     EIBCALEN     = 0
     EIBAID       = X'27'                                    AT   X'20140A2A'
     EIBFN        = X'0000'                                  AT   X'20140A2B'
     EIBRCODE     = X'000000000000'                          AT   X'20140A2D'
     EIBDS        = '........'
 +   EIBREQID     = '........'


 ENTER:    CONTINUE
 PF1 : HELP                PF2 : SWITCH HEX/CHAR     PF3 : END EDF SESSION
 PF4 : SUPPRESS DISPLAYS   PF5 : WORKING STORAGE     PF6 : USER DISPLAY
 PF7 : SCROLL BACK         PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
 PF10: UNDEFINED           PF11: UNDEFINED           PF12: UNDEFINED
```

Figure 6-1 CICS EDF of bank program

4. Follow the prompts for the function keys and their associated actions.

5. When you have finished using CICS EDF, issue the **CEDF <term>,OFF** command to turn it off. In this case, for the current terminal, issue the **CEDF,OFF** command.

6.3.3 Debugging CICS applications using IBM distributed debugger

IBM Rational Developer for AIX and Linux (Rational Developer) is an Eclipse-based software that provides an integrated set of application development tools. Rational Developer provides an integrated development environment (IDE)-based tool facility to develop and debug applications remotely in an AIX or Linux server.

You can also debug TXSeries applications through Rational Developer. You can use the link to configure TXSeries with Rational Developer in a step-by-step manner. For more information, see the following website:

http://www.ibm.com/developerworks/rational/library/debug-mixed-language-cics--txseries/

The IBM Distributed Debugger is a graphical tool that enables you to debug a program. The debugger gives you various options to control the program execution during debugging, such as setting breakpoints, monitoring storage locations, and so on. TXSeries supports IBM distributed debugger to debug CICS-based applications written in IBM COBOL, C, C++, PL/I, and Java languages. IBM Distributed Debugger consists mainly of the following parts:

- The debugger user interface (UI)
- The debug engine

In general, IBM Distributed Debugger is a client/server-based model, where the debug engine acts as a client and the debugger UI acts as the server. This type of client/server-model provides the flexibility of having the debugger UI on one machine, while you are debugging a program that exists and is being executed on another machine. The debug engine and the debug UIs use a TCP/IP link for exchanging data.

With IBM Distributed Debugger you can debug TXSeries applications in one of the following ways:

Local debugging This means that the debugger UI and the debugger engine run on the same machine where the region runs.

Remote debugging This means that the debugger UI is on one machine, and the debug engine is running on another machine where the region runs.

A detailed step-by-step procedure to configure and debug the application using Distributed debugger is documented in the TXSeries documentation library:

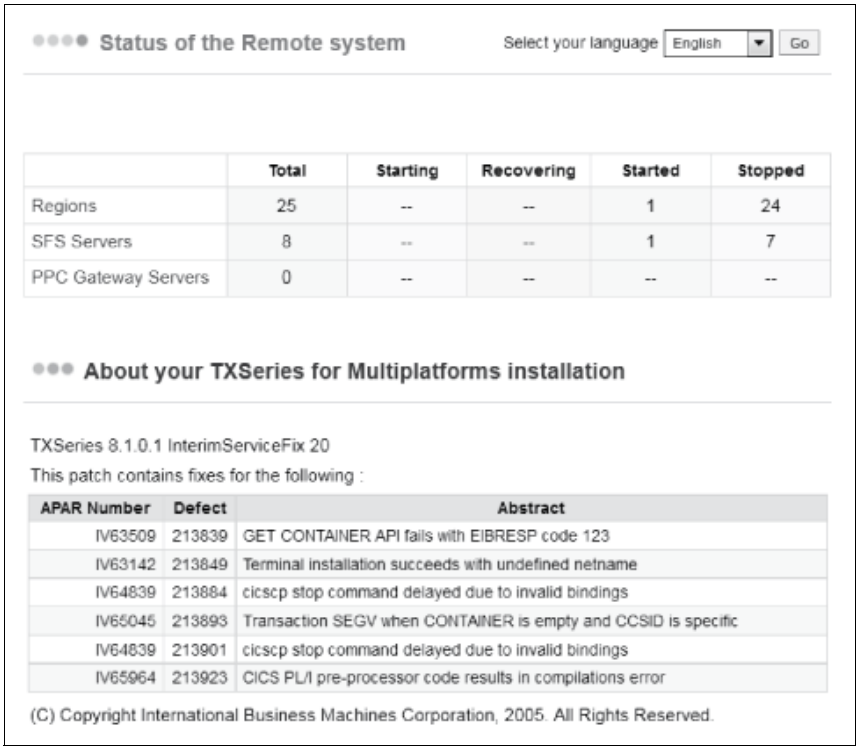
http://www.ibm.com/developerworks/rational/library/debug-mixed-language-cics--txseries/

6.4 Common issues

In this section, we describe the common problems occurring on a TXSeries system, and provide you with a detail analysis for each of the problems.

6.4.1 Determining the installed version of TXSeries

The first page of the TXSeries Administration console shows the exact version of TXSeries, including the authorized program analysis reports (APARs) installed and if you are at an interim fix level. A sample page is shown in Figure 6-2.



Status of the Remote system

Select your language English Go

	Total	Starting	Recovering	Started	Stopped
Regions	25	--	--	1	24
SFS Servers	8	--	--	1	7
PPC Gateway Servers	0	--	--	--	--

About your TXSeries for Multiplatforms installation

TXSeries 8.1.0.1 InterimServiceFix 20

This patch contains fixes for the following :

APAR Number	Defect	Abstract
IV63509	213839	GET CONTAINER API fails with EIBRESP code 123
IV63142	213849	Terminal installation succeeds with undefined netname
IV64839	213884	cicscp stop command delayed due to invalid bindings
IV65045	213893	Transaction SEGV when CONTAINER is empty and CCSID is specific
IV64839	213901	cicscp stop command delayed due to invalid bindings
IV65964	213923	CICS PL/I pre-processor code results in compilations error

(C) Copyright International Business Machines Corporation, 2005. All Rights Reserved.

Figure 6-2 Sample page: Interim fix level

To determine the version from a command-line interface (CLI), issue the **`cicscp version`** command:

```
# cicscp version
TXSeries 8.1.0.0 s810-L131118
```

If you have installed a particular fix pack, test fix, or interim fix of TXSeries, you can determine if a particular APAR is present by checking the following files:

1. The `<CICS Install directory>/etc/Version` directory contains the version of TXSeries, and the list of APARs installed as part of the interim fix or test fix. For example, on AIX, it is the `/usr/lpp/cics/etc/Version` directory.
2. The `<CICS Install directory>/etc/APARs.txt` file contains the list of APARs currently included in the installed fix pack. For example, on AIX, it is the `/usr/lpp/cics/etc/APARs.txt` file.

Alternatively, the region's `console.nnnnnn` log can be used to know the TXSeries version. The console log shows the first few lines of the `console.nnnnnn` log, showing you the major release, minor release, and the fix pack level of the TXSeries installed. Example 6-2 shows an example of the console log.

Example 6-2 Console log

```
ERZ010190I/0020 10/08/14 07:46:07.621596000 txdemo 691/0001 : CICS Rele
ase Number 8.1.0.0 Revision Level TXSeries 8.1.0.1 s810-L140620
ERZ010135I/0362 10/08/14 07:46:07.621745000 txdemo 691/0001 : CICS regi
on 'txdemo' is being started with locale categories 'en_US'
```

6.4.2 Analyzing ASRA abends

`ASRA` abends are caused by faults within the CICS application module. The faults can be an exception raised (on Windows), or a signal generated (on UNIX and Linux) by the application code.

> **Important:** The CICS application modules are not just apps that have **`EXEC CICS`** commands in them. They can include other modules, such as **`EXEC SQL`**, IBM WebSphere MQ calls, or any other IBM or third-party product modules. Therefore, it is important to note that when an `ASRA` abend is generated, an exception or a signal is generated outside of the TXSeries CICS code.

The first source of information that you must analyze for these abends is the traceback files. These traceback files contain the stack of the faulty application through which you can determine where the exception or the signal is generated. The traceback file also contains other general information:

- When the exception or the signal is generated
- What signal is generated
- Which transaction and program caused the fault

To determine the faulty code in the application module, you can use the Offset information provided in the traceback file.

Example 6-3 shows you a snapshot of the traceback file.

Example 6-3 Snapshot of the traceback file

```
*********************** TraceBack Details *************************

>>>>>>>>>>>>>>>>>>>>>>>> TraceBack Header  <<<<<<<<<<<<<<<<<<<<<<<<<<
TIMESTAMP : 11/08/05 15:32:53.596396452
REGION    : hostreg
TRANID    : CECI
PROGRAM   : CREDITAP
SRVID     : 102
PID       : 843886
TID       : 1
SIGNAL    : 11
>>>>>>>>>>>>>>>>>>>>>>>> Function Stack    <<<<<<<<<<<<<<<<<<<<<<<<<<
22 - Function strncpy Offset = 00F4
21 - Function main Offset = 01AC
20 - Function PinCA_StartC Offset = 01CC
19 - Function TasPR_CallApplication Offset = 0318
18 - Function TasPR_RunProgram Offset = 1CA4
17 - Function TasPR_IRun Offset = 20A4
16 - Function TasPR_Run Offset = 1428
15 - Function PinCA_Route Offset = 09CC
14 - Function PinCA_Run_UM Offset = 00E8
13 - Function PinCI_Execute Offset = 0794
12 - Function PinCI_CECIS Offset = 06EC
11 - Function main Offset = 01E0
10 - Function PinCA_StartC Offset = 01CC
9 - Function TasPR_CallApplication Offset = 0318
8 - Function TasPR_RunProgram Offset = 1CA4
7 - Function TasPR_IRun Offset = 20A4
6 - Function TasTA_Exec Offset = 1F2C
5 - Function ComCR_CRTEBackEnd Offset = 0D94
4 - Function ComCR_CRTE Offset = 0174
3 - Function TasTA_Exec Offset = 1E78
2 - Function TasTA_Run Offset = 1BF0
1 - Function main Offset = 0C20
0 - Function __start Offset = 0088
>>>>>>>>>>>>>>>>>>>>>>>> Registers Dump    <<<<<<<<<<<<<<<<<<<<<<<<<<
GPR00 = 00000000 ---  GPR01 = 2FF17280 ---  GPR02 = F03F2A40 ---
 GPR03 = 2FF172E8 ---  GPR04 = FEFFFFFF ---  GPR05 = 2FF172E7 ---
 GPR06 = 0000001C ---  GPR07 = 00000000 ---  GPR08 = 0001CBE9 ---
 GPR09 = 00000000 ---  GPR10 = 00000000 ---  GPR11 = 000034E0 ---
```

However, sometimes you see a traceback file like the one shown in Example 6-4. Here the function entries are indicated as `NULL`.

Example 6-4 Traceback file with NULL function entries

```
>>>>>>>>>>>>>>>>>>>>>>>> TraceBack Header  <<<<<<<<<<<<<<<<<<<<<<<<<

TIMESTAMP : 11/09/14 16:36:09.985128249
REGION    : txdemo
TRANID    : CPMI
PROGRAM   : CUSTECI
SRVID     : 101
PID       : 1274098
TID       : 1
SIGNAL    : 11

>>>>>>>>>>>>>>>>>>>>>>>> Function Stack    <<<<<<<<<<<<<<<<<<<<<<<<<

0 - Function (NULL) detected
0 - Function (NULL) detected
0 - Function (NULL) detected
0 - Function (NULL) detected
0 - Function (NULL) detected
0 - Function (NULL) detected
10 - Function PinCA_StartIBMCob Offset = 08D8
9 - Function TasPR_CallApplication Offset = 0318
8 - Function TasPR_RunProgram Offset = 1CA4
7 - Function TasPR_IRun Offset = 20A4
6 - Function TasPR_Run Offset = 1428
5 - Function PinCA_Route Offset = 09CC
4 - Function ComFS_APPCServ Offset = 0CB8
3 - Function TasTA_Exec Offset = 1E10
2 - Function TasTA_Run Offset = 1BF0
1 - Function main Offset = 0C20
0 - Function __start Offset = 0088

>>>>>>>>>>>>>>>>>>>>>>>> Registers Dump    <<<<<<<<<<<<<<<<<<<<<<<<<

 GPR00 = 00000031 ---  GPR01 = 2FF1C9B0 ---  GPR02 = F0EF7778 ---
 GPR03 = 00000000 ---  GPR04 = 205D23E5 ---  GPR05 = 00000028 ---
 GPR06 = 2050E10B ---  GPR07 = 2050E109 ---  GPR08 = F0EFB7E4 ---
```

This incorrect traceback indicates that the debug information is missing in the concerned module. To get a proper traceback, you must have a debug version of the modules, and the dependent modules.

6.4.3 Analyzing A147 abends

A147 abends are not caused by just one reason. TXSeries reports A147 abends for multiple cases, for some of the following reasons:

- Unexpected termination of application server processes

  This can be due to an `exit` or `_exit` system call, or a **`STOP RUN`** command in a COBOL program made in the application module. Alternatively, this can be from an IBM or third-party library loaded in the application server process. It can be a signal that is unhandled, or can be due to unrecoverable system process issues.

  In some of the cases, a *core* file (On UNIX and Linux) is generated in the region's `dumps` directory. Analyze this file to determine the cause.

- **`PURGE`** or **`FORCEPURGE`** of a task

  This can be due to the initiation of a **`CEMT I TASK() PURGE`** or **`CEMT I TASK() FORCEPURGE`** command. It is the same as when doing it using **`EXEC CICS SET TASK()`** API. When TXSeries detects that a client has disconnected abnormally, it automatically initiates a **`FORCEPURGE`**, and an A147 abend is reported.

  We can determine if the task is purged or force-purged by looking at the messages in the `console.`*`nnnnnn`* log.

- Transaction or client timeout

  This is initiated through the `timeout` attribute set in the TD stanza or due to a timeout set in the external call interface (ECI) client calls.

7

Upgrading to the latest version of TXSeries

This chapter describes how to migrate your existing IBM TXSeries Customer Information Control System (IBM CICS) system to a newer version of IBM TXSeries for Multiplatforms.

This chapter provides information about the following topics:

7.1 Migration guidelines

The migration procedure varies from one environment to another, depending on various system components and external modules that are connected to the TXSeries product. However, using the BigBlueBank system, we give you a basic idea of the migration procedure.

The first and the most important step to migrate any product to a newer version is to review the release notes. You have to understand the various changes that are made in the new release, and see how that helps or affects the system environment after migration.

The following list describes some of the important issues that you must consider before migration:

- Supported versions of IBM and third-party products
- New or changed features
- Compatibility with earlier versions for CICS server applications
- CICS application programming interface (API) and system programming interface (SPI) compatibility, changes, and behavior
- Product environment variables
- Known limitations and defects

For more information about the new and changed features in TXSeries, see the following website:

http://www.ibm.com/support/knowledgecenter/SSAL2T/welcome

7.2 Migrating BigBlueBank system

The BigBlueBank system is developed and run under a TXSeries CICS 5.1 environment. We demonstrate the step-by-step procedure on how to migrate the current system to the environment of the latest version of TXSeries for Multiplatforms. The following components are identified in the BigBlueBank system for this migration:

- Resource manager: Database systems, IBM DB2
- Resource manager: Messaging systems, IBM WebSphere MQ
- File server: Structured File Server (SFS)
- Language runtimes: C, C++, Java, IBM COBOL
- Batch applications: External File Handler (EXTFH) support
- Client systems
- Utility programs: Console (monitor) application

7.2.1 First steps of migration

As a preliminary step before starting the migration, it is important to complete the following tasks:

- Review the release notes:
  - Check for any change in the environment variable.
  - Check if you have to recompile your applications.
- Understand the prerequisites and dependencies of the newer environment:
  - Hardware requirements
  - Memory and space requirements
  - Software requirements for IBM and third-party product software
- Set up a machine with the necessary hardware and software prerequisites.

Start the migration activity of the BigBlueBank system as described in the following section.

7.2.2 Preparing the systems for migration

The BigBlueBank system currently runs on PONS1 (what we call a *source system*) with TXSeries for Multiplatforms V5.1 environment, and PONS2 (what we call a *target system*), which has a TXSeries for Multiplatforms V6.0 environment. Both PONS1 and PONS2 are IBM pSeries machines running AIX V5.3 operating system.

PONS1 (source system)

On the source system, we perform the following steps:

1. Stop the TXSeries CICS regions `txdemo` running the BigBlueBank system.
2. Ensure that the SFS, IBM DB2 instance is running.
3. Stop the Systems Network Architecture (SNA) nodes.

PONS2 (target system)

On the target machine, we install and configure the following products:

- DB2:
  - Create an instance named `db2local` and a database `cicstest`.
  - On the `cicstest` database, grant `select` privilege on `dba_pending_transactions` to user `cics`.
  - Add user `db2local` to the `cics` group.

- IBM WebSphere MQ:
  - Create a queue manager with the name `mqdemo`.
  - Add `mqm` to the `cics` group.
  - Create an operating system (OS) user called `CICSUSER` and add this to the `mqm` group.
  - Add the system user (who will be creating and managing WebSphere MQ queues) to mqm group.
- TXSeries for Multiplatforms V6.0:
  - Create an SFS with the host name (`PONS2`).
  - Create and start a test CICS region.
  - Add CICS and SFS users to the `cics` group.
- IBM CICS Transaction Gateway (CICS TG)
- IBM Communications Server for AIX
- IBM Directory Server
- IBM COBOL for AIX

7.2.3 Migrating File Manager (SFS) data

We use SFS as the File Manager for the BigBlueBank system. It has the following Virtual Storage Access Method (VSAM) data files:

- `BANKACCT`
- `BANKCUST`
- `BANKERR`

This section shows you how to migrate these VSAM data files to a target system.

PONS1

We set the `ENCINA_BINDING_FILE` to the location of the `server_bindings` file, which is required to access the SFS. We export the VSAM data files to a flat file on the operating system's file system using the **`cicssdt`** tool. The tool exports both the VSAM data file definition and all of its records on to the flat file.

> **Tip:** You can use the **`cicsddt`** (DB2) and **`cicsodt`** (Oracle) tools if your VSAM data files are under either of those databases.

To export the VSAM data files `BANKACCT`, `BANKCUST` and `BANKERR`, use the **`stof`** command to the flat files `BANKACCT.sdt`, `BANKCUST.sdt`, and `BANKERR.sdt`.

Example 7-1 on page 363 shows the command to export these VSAM data files.

Example 7-1 Exporting the VSAM files

```
{pons1:root}/ ->
{pons1:root}/ -> set
ENCINA_BINDING_FILE=/var/cics_servers/server_bindings
{pons1:root}/ -> echo $ENCINA_BINDING_FILE
/var/cics_servers/server_bindings
{pons1:root}/ -> cicssdt -s /.:/cics/sfs/pons1

ERZ037068I/0500: CICSSDT talking to: '/.:/cics/sfs/pons1'
ERZ037069I/0501: Version [2.1 : 12-07-1995]
ERZ037070I/0502: Contacting Server ...
ERZ037103I/0704: (Server OK)
cicssdt: -> stof BANKACCT

[Flat Filename ..............: BANKACCT.sdt
[Dump file Asis ? ..... Y/[N]: N
[Dump Record(s) .............: ALL

[Dumping All Records]
[Dumping Record #][00000035]
cicssdt: -> stof BANKCUST

[Flat Filename ..............: BANKCUST.sdt
[Dump file Asis ? ..... Y/[N]: N
[Dump Record(s) .............: ALL

[Dumping All Records]
[Dumping Record #][00000020]
cicssdt: -> stof BANKERR

[Flat Filename ..............: BANKERR.sdt
[Dump file Asis ? ..... Y/[N]: N
[Dump Record(s) .............: ALL

[Dumping All Records]
[Dumping Record #][00000047]
cicssdt: -> quit

{pons1:root}/ -> ls -l BANK*.sdt
-rw-r--r--   1 root     system         3366 Nov 13 14:23 BANKACCT.sdt
-rw-r--r--   1 root     system         6572 Nov 13 14:23 BANKCUST.sdt
-rw-r--r--   1 root     system         4301 Nov 13 14:23 BANKERR.sdt
{pons1:root}/ ->
```

PONS2

We transfer the flat files `BANKACCT.sdt`, `BANKCUST.sdt`, and `BANKERR.sdt` to the target machine: PONS2. Example 7-2 shows the command to transfer the files.

Example 7-2 Transferring the files to the target machine

```
{pons1:root}/ -> ftp pons2
Connected to pons2.
220 pons2 FTP server (Version 4.2 Thu Jul 21 05:28:02 CDT 2005) ready.
Name (pons2:root): root
331 Password required for root.
Password:
230-Last unsuccessful login: Sun Nov 13 15:02:20 NFT 2005 on ftp from
pons1
230-Last login: Sun Nov 13 15:02:26 NFT 2005 on ftp from pons1
230 User root logged in.
ftp> cd /var/cics51_data/SFS
250 CWD command successful.
ftp> bin
200 Type set to I.
ftp> verbose
Verbose mode off.
ftp> put BANKACCT.sdt
ftp> put BANKCUST.sdt
ftp> put BANKERR.sdt
ftp> ls
BANKACCT.sdt
BANKCUST.sdt
BANKERR.sdt
ftp> quit
{pons1:root}/ ->
```

Note: The `ENCINA_BINDING_FILE` setting is not required to be set under TXSeries for Multiplatforms V6.0.

We import the transferred flat file using the **ftos** command to the SFS running on PONS2. Example 7-3 shows how to import the BANKERR VSAM data file.

Example 7-3 Importing the files

```
{pons2:root}/ -> cicssdt -s /.:/cics/sfs/pons2

ERZ037068I/0500: CICSSDT talking to: '/.:/cics/sfs/pons2'
ERZ037069I/0501: Version [2.1 : 12-07-1995]
ERZ037070I/0502: Contacting Server ...
ERZ037103I/0704: (Server OK)
cicssdt: -> ftos BANKERR

[Flat Filename ..............: /var/cics51_data/SFS/BANKERR.sdt
ERZ037129E/0901: Failed to open file (Server: '/.:/cics/sfs/pons2'
File: 'BANKERR' Error: 'SFS_NO_SUCH_FILE')
[File "BANKERR" doesn't exist. Create file ? [Y]/N Y

[SFS Server Volume Name .....: sfs_Spons2
[File Organisation [E/R/B] ..: B[treeClustered]
[Field 01: Name .............: ERROR_NO
[Field 01: Type .............: byteArray
[Field 01: Size .............: 5
[Field 02: Name .............: ERROR_TEXT
[Field 02: Type .............: byteArray
[Field 02: Size .............: 72
[Field 03: Name .............: FILLER
[Field 03: Type .............: byteArray
[Field 03: Size .............: 3
[Field 04: Name .............:
[Maximum Number Of Records ..: SFS_NATURAL_RECORD_LIMIT
[Primary Index Name .........: ERROR_NO
[Is Index Unique ? [Y]/N ....: Y
[Index Field 01: Field Name .: ERROR_NO
[Index Field 01: Ordering ...: a[scending]
[Index Field 02: Field Name .:
[Number Of Pages To Allocate : 1

[Loading Record #][00000047]
cicssdt: -> quit

{pons2:root}/ ->
```

Similarly, import the BANKCUST and BANKACCT VSAM data files as well. This completes the migration of the VSAM data files on SFS.

7.2.4 Migrating resource manager (IBM DB2) data

We use DB2 for logging audit trail messages for the BigBlueBank system. We have the CUSTAUDIT tables present in the DB2 database. In this section, we show you how to migrate the DB2 table to a target system.

PONS1

The DB2 instance must be running on the PONS1 machine. We log on to the DB2 instance user, and connect to the DB2 database. The DB2 instance we use is `db2local`, and the database we use is `cicstest`. We export the CUSTAUDIT table in the database using the DB2 **`EXPORT`** command. This command writes the records of the table to a flat file named `CUSTAUDIT.ixf` on the operating system's file system. Example 7-4 shows the commands to export the table.

Example 7-4 Export table commands

```
{pons1:root}/ -> su – db2local
$
$ echo $DB2INSTANCE
db2local
$
$ db2 connect to cicstest

Database Connection Information
 Database server        = DB2/6000 8.2.0
 SQL authorization ID   = DB2LOCAL
 Local database alias   = CICSTEST

$
$ db2 “SELECT COUNT(*) FROM CUSTAUDIT”
1
37

1 record(s) selected.
$
$ db2 export to CUSTAUDIT.ixf of ixf messages msgs.txt “SELECT * FROM
CUSTAUDIT”

Number of rows exported: 37
$ ls –l CUSTAUDIT.ixf
-rw-r-----   1 db2local guest          8136 Nov 13 14:41 CUSTAUDIT.ixf
$
$ exit
{pons1:root}/ ->
```

PONS2

We create the CUSTAUDIT table definition in the database using the DB2 **CREATE** command. We transfer the flat file `CUSTAUDIT.ixf` on to the target machine using the File Transfer Protocol (FTP) program. Example 7-5 shows the command that we run on the target machine to import the DB2 data.

Example 7-5 Commands to import DB2 data

```
$ pwd
/var/cics51_data/DB2
$ echo $DB2INSTANCE
db2local
$ db2 connect to cicstest

   Database Connection Information

 Database server        = DB2/6000 8.2.0
 SQL authorization ID   = DB2LOCAL
 Local database alias   = CICSTEST

$ db2 “create table CUSTAUDIT (…............)”

$ db2 IMPORT FROM CUSTAUDIT.ixf OF IXF MESSAGES msgs.txt INSERT INTO
CUSTAUDIT

Number of rows read         = 37
Number of rows skipped      = 0
Number of rows inserted     = 37
Number of rows updated      = 0
Number of rows rejected     = 0
Number of rows committed    = 37

$ db2 “SELECT COUNT(*) FROM CUSTAUDIT”

1
         37

  1 record(s) selected.

$ exit
{pons2:root}/var ->
```

We use the DB2 **SELECT** statement to confirm that the number of records imported is correct. This completes the migration of the table on the DB2 database.

7.2.5 Configuring resource manager (WebSphere MQ)

We use WebSphere MQ to send notifications to clients of the BigBlueBank system when their respective savings account is updated. We do not have any data stored on the WebSphere MQ system. Therefore, we only create the necessary LOCAL QUEUE definition on the target machine, as shown in Example 7-6.

Example 7-6 Creating the local queue

```
{pons2:root}/tmp -> su - mqm
$
$ runmqsc mqdemo
5724-B41 (C) Copyright IBM Corp. 1994, 2002.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager mqdemo.


DEFINE QLOCAL (CICS.LOCAL.QUEUE) DESCR('Local Queue') PUT(ENABLED)
GET(ENABLED) PROCESS(CICS.PROCESS) REPLACE
     1 : DEFINE QLOCAL (CICS.LOCAL.QUEUE) DESCR('Local Queue')
PUT(ENABLED) GET(ENABLED) PROCESS(CICS.PROCESS) REPLACE
AMQ8006: WebSphere MQ queue created.
END
     2 : END
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
$
```

7.2.6 Exporting and importing the region data

This section is required when migrating regions for systems. However, if you are migrating on the same system, you can skip this section and proceed to 7.2.7, “Migrating CICS regions” on page 370.

The BigBlueBank system runs on a TXSeries CICS region called *txdemo*. This section explains the steps to migrate these two regions on to the target machine.

PONS1

We use the **cicsexport** command to prepare the region for migration. The command generates an archive file, which contains the region's data. Example 7-7 shows the txdemo regions that are prepared for the migration.

Example 7-7 Preparing regions for migration

```
{pons1:root}/ ->
{pons1:root}/ -> cicsexport -r txdemo -o txdemo.repos -t /tmp
ERZ046338I/0420: Successfully exported region 'txdemo'
{pons1:root}/ -> ls -l txdemo.repos
-rw-r--r--   1 cics     cics        2480386 Nov 13 14:51 txdemo.repos
{pons1:root}/ ->
```

The **cicsexport** command creates an archive file, `txdemo.report`, of the txdemo region. We transfer the archive files to the target machine using the FTP program.

PONS2

We use the **cicsimport** command to import the region's data from the transferred archive file. Example 7-8 shows the commands to import the txdemo region.

Example 7-8 Commands to import txdemo region

```
{pons2:root}/var/cics51_data/REGION -> ls *repos
txdemo.repos
{pons2:root}/var/cics51_data/REGION -> cicsimport -r txdemo -i
txdemo.repos
ERZ046348I/0427: Creating region 'txdemo' from archive
'/var/cics51_data/REGION/txdemo.repos'
.......................................................................
.......................................................................
<ERZ warning messages for unsupported attributes from resource
definitions during migration>
.......................................................................
.......................................................................
ERZ010114I/0728: Creating subsystem 'cics.txdemo' for region 'txdemo'
ERZ046339I/0421: Successfully imported region 'txdemo'
{pons2:root}/var/cics51_data/REGION ->
```

Warning messages are shown indicating that the region resource definition attributes are not being supported. This is due to the removal or change in resource definition attributes across the different versions of TXSeries.

7.2.7 Migrating CICS regions

We have imported both the regions successfully. Now we migrate the region using the **cicsmigrate** tool. This tool generates a migration shell script for the given region. Example 7-9 shows how we generate the **txdemo.sh** script using the **cicsmigrate** tool for the txdemo region. We run the **txdemo.sh** script to perform the migration.

Example 7-9 Generating the txdemo.sh script

```
{pons2:root}/var/cics51_data/REGION ->
{pons2:root}/var/cics51_data/REGION -> cicsmigrate -?
ERZ046141I/0450: Usage: cicsmigrate {-?|[ -r regionName | -s | -c | -p
| -m ]
–g filename –o logFileName –d}
{pons2:root}/var/cics51_data/REGION -> cicsmigrate –r txdemo –g
txdemo.sh –o txdemo.log
{pons2:root}/var/cics51_data/REGION ->
{pons2:root}/var/cics51_data/REGION -> ls –l txdemo.sh
-rw-rw---- 1 root system 5566 Nov 13 17:00 txdemo.sh
{pons2:root}/var/cics51_data/REGION ->
{pons2:root}/var/cics51_data/REGION -> ksh txdemo.sh
{pons2:root}/var/cics51_data/REGION -> ls –l txdemo.log
-rw-r–r–1 root system 13517 Nov 13 17:00 txdemo.log
{pons2:root}/var/cics51_data/REGION -> tail txdemo.log

……….
……….
<Content of txdemo.log indicating the steps and results of migration,
related to resource definitions of the region>
……….
……….

{pons2:root}/var/cics51_data/REGION ->
```

Tip: The txdemo.log contains the result of the txdemo region migration.

7.2.8 Configuring region environment and resource definition files

We have successfully migrated the data and the CICS regions on the target system. We now review the region resource definitions for necessary updates. This is required because some of the resource definition attributes still have references to the source system, and are not valid on the target system.

The region's `environment` and `region.properties` files require updates if they are different from the source system. The BigBlueBank system uses the same contents for these files as on the source system. Therefore, we do not update these files on the target system.

> **Important:** The region's environment and region properties files sometimes contain path name, which you have to update on the target system, if they are different from the source system.

Communication definitions (CD)

The BigBlueBank system has two TXSeries CICS regions, which are interconnected through `cics_tcp` intersystem communication (ISC) facility. Therefore, the CD resource definition has reference to the source machine in the `RemoteTCPAddress` attribute. We change the value of this attribute from `PONS1` to `PONS2`.

> **Tip:** If the remote system is on the same machine, setting `RemoteTCPAddress=""` sets it to the local host by default. Setting the attribute this way does not necessitate changes to it during migration.

File definitions (FD)

We have to change the `FileServer` attribute of all of the VSAM FDs. This attribute contains the name of the SFS on the source system, which is not valid in the target system. We now change the attributes for the `CUSTMAST`, `ERRFILE`, and `ACCMAST` FDs.

Example 7-10 shows the commands to update the FileServer attribute.

Example 7-10 Commands to update the FIleServer

```
{pons2:root}/ -> cicsget -c fd -r txdemo -l
CUSTMAST Customer Master File
ERRFILE Error Message File
ACCMAST Account Master File
{pons2:root}/ -> cicsget -c fd -r txdemo CUSTMAST | grep FileServer
FileServer="/.:/cics/sfs/pons1"
{pons2:root}/ -> cicsupdate -c fd -r txdemo CUSTMAST \
> FileServer="/.:/cics/sfs/pons2"
{pons2:root}/ -> cicsget -c fd -r txdemo CUSTMAST | grep FileServer
FileServer="/.:/cics/sfs/pons2"
{pons2:root}/ ->
{pons2:root}/ -> cicsget -c fd -r txdemo ERRFILE | grep FileServer
FileServer="/.:/cics/sfs/pons1"
{pons2:root}/ -> cicsupdate -c fd -r txdemo ERRFILE \
> FileServer="/.:/cics/sfs/pons2"
{pons2:root}/ -> cicsget -c fd -r txdemo ERRFILE | grep FileServer
FileServer="/.:/cics/sfs/pons2"
{pons2:root}/ ->
{pons2:root}/ -> cicsget -c fd -r txdemo ACCMAST | grep FileServer
FileServer="/.:/cics/sfs/pons1"
{pons2:root}/ -> cicsupdate -c fd -r txdemo ACCMAST \
> FileServer="/.:/cics/sfs/pons2"
pons2:root}/ -> cicsget -c fd -r txdemo ACCMAST | grep FileServer
FileServer="/.:/cics/sfs/pons2"
{pons2:root}/ ->
```

Tip: If the file server name is the same as the host name, set the attribute to `FileServer="/.:/cics/sfs/%S"` to avoid needing changes during migration.

Listener definitions (LD)

For LDs that are defined for Transmission Control Protocol/Internet Protocol (TCP/IP), we check the `TCPAddress` attribute, and update it with the target machine's TCP address. We also add the relevant entries in `/etc/services` to reflect the value of the `TCPService` attribute. Example 7-11 shows the command to update the `TCPService` attribute.

Example 7-11 TCPService attribute

```
{pons2:root}/tmp -> cat /etc/services | grep BigBlueBank
hostsvc         20001/tcp       # BigBlueBank system
{pons2:root}/tmp ->
```

Monitoring definitions (MD)

We have to update the `UserMonitorModule` attribute with the location of the customized user-monitoring module. The BigBlueBank system stores the user monitoring module in the region's bin directory. Therefore, we do not update the path name in the attribute. TXSeries CICS, by default, searches the user-monitoring module under the respective region's bin directory.

Program definitions (PD)

We have to update the `PathName` attribute with the location of the maps, server, and user-exit programs. The BigBlueBank system stores the maps, server, and user-exit programs on the same location as that of the source system. Therefore, we do not update the `PathName` in the PD definitions.

Region definitions (RD)

The RD has several attributes that require updates:

RDBMSInstance
This attribute is valid when we use DB2 or Oracle as the File Manager. The respective database name requires update if it is different from the source system. The BigBlueBank system uses SFS as its File Manager, therefore no update is required for this attribute.

ESMModule
The location of the external security manager (ESM) module requires update if it is different from the source system. The BigBlueBank system does not use the ESM feature, therefore no update is required for this attribute.

EAMModule
The location of the external authentication manager (EAM) module requires update if it is different from the source system. The BigBlueBank system stores the EAM module on the same location as that of the source system, therefore we do not update this attribute.

LocalSysId/LocalNetworkName
The values of these attributes require update if they are different from the source machine. The BigBlueBank system uses the default values, therefore we do not update these attributes.

DumpName/CoreDumpName/TraceDirectorySystem/TraceDirectoryUser/StatFile
The values of these attributes require update if they are different from the source system. The BigBlueBank system uses the default values, therefore we do not update these attributes.

DefaultFileServer The value of this attribute requires update if it is different from the source system. The BigBlueBank system uses the short-form notation to refer to the file server, therefore we do not update this attribute.

IIOPGroupHost/IIOPGroupPort/IIOPGroupUUID/IIOPGroupSSLPort

The values of these attributes require update if they are different from the source system. The BigBlueBank system uses the default values, therefore we do not update these attributes.

Transient data definitions (TDD)

The value of the `ExtrapartitionFile` attribute requires update if it is different from the source system. The BigBlueBank system does not use transient data queues (TDQ), therefore we do not update this attribute.

Extended architecture resource manager definitions (XAD)

The value of the `XAOpen` attribute requires update if it is different from the source system. The location of the switch-load file also requires update in the `SwitchLoadFile` attribute. The BigBlueBank system uses the `XAOpen` string, and the location of `SwitchLoadFile` as used in the source system. Therefore we do not update these attributes.

Important: You must review the release notes to check whether there are any new attributes introduced in the resource definition files (stanzas).

7.2.9 Preparing the switch-load module

In BigBlueBank system, we use DB2 and WebSphere MQ as resource managers (RMs). We show you how to build the switch-load modules for these RMs.

Preparing the switch-load module for DB2

TXSeries for Multiplatforms supplies the switch-load module for DB2. The switch-load file is named cicsxadb2. It is in the `<`*`TXSeries installation directory`*`>/bin` directory. We use this switch-load file for the BigBlueBank system.

Preparing the switch-load module for WebSphere MQ

We use the command shown in Example 7-12 to build the switch-load file for WebSphere MQ.

Example 7-12 Building the switch-load file for WebSphere MQ

```
{pons2:root}/tmp -> export CICSREGION=txdemo
{pons2:root}/tmp -> echo "amqzscix" > tmp.exp
{pons2:root}/tmp -> xlc_r $(MQM_ROOT)/samp/amqzscix.c \
> -I/usr/lpp/cics/include –I$(MQM_ROOT)/inc \
> -e amqzscix -bE:tmp.exp -bM:SRE \
> -o /var/cics_regions/$CICSREGION/bin/amqzscix \
> /usr/lpp/cics/lib/regxa_swxa.o –L$(MQM_ROOT)/lib \
> -L/usr/lpp/cics/lib -lcicsrt -lEncina \
> -lsarpc -lmqmcics_r -lmqmxa_r -lmqzi_r \
> -lmqmcs_r -lmqmzse
{pons2:root}/tmp -> rm tmp.exp
{pons2:root}/tmp -> ls -l /var/cics_regions/txdemo/bin/amqzscix
-rwxr-xr-x 1 root system 66762 Nov 14 11:08
/var/cics_regions/txdemo/bin/amqzscix
{pons2:root}/tmp ->
```

7.2.10 External modules

The BigBlueBank systems use the user exit, external authentication, and monitoring modules. We use FTP to transfer these modules from the source system to the target system.

7.2.11 Preparing language run time

In the BigBlueBank system, we use these language runtimes: IBM COBOL, C, C++, and Java. For IBM COBOL, C, C++, and Java, the language run times are supplied with the TXSeries product. Therefore, there is no action required.

7.2.12 Batch applications

The BigBlueBank system has certain batch applications that use the EXTFH support. We use FTP to transfer these binary modules from the source system to the target system. When you transfer binary modules from one system to another, you must make sure that the target OS enables compatibility for running the binary modules generated in the source OS.

7.2.13 CICS applications

The BigBlueBank system has IBM COBOL-based, C-based, C++-based, and Java-based CICS server applications. In the new release, we check whether there are changes in syntax or behavior for the CICS APIs that we use in our applications. We do this syntax check by running the **cicstran** command on all of our source applications.

Because there are no changes in the API that we use in the BigBlueBank system, we decided not to recompile these applications. Therefore, we transfer (using FTP) the binary applications from the source system to the target system as is.

We use the **cicstcl** command to translate, compile, and link-edit the BigBlueBank system applications. However, you must review the compiler/link options passed by TXSeries if you start the compiler/linker separately. The **cicstcl** command output shows the default options that must be passed to the compiler/linker for building TXSeries applications.

Requirement: If there are any IBM or third-party product interfaces within the application, it is mandatory to check the relevant product release notes to verify whether you are required to recompile the application on the target environment.

7.2.14 Configuring the client

For the BigBlueBank system, we have to configure the CICS TG product to connect to the CICS region. We use the server details shown in Example 7-13 to configure CICS TG on the target system.

Example 7-13 Configuring CICS Transaction Gateway

```
SECTION SERVER = TXDEMO
DESCRIPTION=TXDEMO TXSeries Region
UPPERCASESECURITY=N
PROTOCOL=TCPIP
NETNAME=pons2
PORT=1435
CONNECTTIMEOUT=0
TCPKEEPALIVE=N
ENDSEC-TION
```

Important: You must ensure that the ports used by your client applications do *not* conflict with other software running on your target system.

7.2.15 Starting the TXSeries CICS regions

We have completed the migration of the BigBlueBank system from the TXSeries for Multiplatforms V5.1 environment to the current version of TXSeries for Multiplatforms environment. We start the migrated CICS region, as shown in Example 7-14.

Example 7-14 Starting the CICS region

```
{pons2:root}/ -> cicscp –v start region txdemo StartType=cold
ERZ096122I/0264: Processing a ‘start region’ command
ERZ096158I/0264: Starting region ‘txdemo’
ERZ096111I/0224: Processing a start sfs_server command
ERZ096141I/0224: Starting SFS server ‘/.:/cics/sfs/pons2’
ERZ096112I/0229: The SFS server ‘/.:/cics/sfs/pons2’ is already running
ERZ096113I/0231: SFS server ‘/.:/cics/sfs/pons2’ successfully started
ERZ096111I/0224: Processing a start sfs_server command
ERZ096141I/0224: Starting SFS server ‘/.:/cics/sfs/pons2’
ERZ096112I/0229: The SFS server ‘/.:/cics/sfs/pons2’ is already running
ERZ096113I/0231: SFS server ‘/.:/cics/sfs/pons2’ successfully started
ERZ038176I/0339: Adding TSQ file ‘txdemocicsrectsqfile’ to server
‘/.:/cics/sfs/pons2’, volume ‘sfs_Spons2’.
ERZ038176I/0344: Adding TSQ file ‘txdemocicsnrectsqfil’ to server
‘/.:/cics/sfs/pons2’, volume ‘sfs_Spons2’.
ERZ038177I/0349: Adding TDQ file ‘txdemocicstdqlgfile’ to server
‘/.:/cics/sfs/pons2’, volume ‘sfs_Spons2’.
ERZ038177I/0354: Adding TDQ file ‘txdemocicstdqphfile’ to server
‘/.:/cics/sfs/pons2’, volume ‘sfs_Spons2’.
ERZ038177I/0359: Adding TDQ file ‘txdemocicstdqnofile’ to server
‘/.:/cics/sfs/pons2’, volume ‘sfs_Spons2’.
ERZ038178I/0364: Adding Local Queueing file ‘txdemocicsnlqfile’ to
server ‘/.:/cics/sfs/pons2’, volume ‘sfs_Spons2’.
ERZ038178I/0369: Adding Local Queueing file ‘txdemocicsplqfile’ to
server ‘/.:/cics/sfs/pons2’, volume ‘sfs_Spons2’.
ERZ010013I/0024: CICS has removed the lock file for region ‘txdemo’
ERZ096126I/0276: Region ‘txdemo’ successfully started
ERZ096002I/0003: cicscp command completed successfully
{pons2:root}/ ->
```

7.2.16 Problems during migration

After completing the migration steps mentioned previously, the CICS region did not start. For more details about the problems that we encountered during migration and their solutions, see 7.4, “Migration issues with BigBlueBank” on page 379.

7.2.17 Connecting to BigBlueBank

We have now successfully started the CICS region, and we connect to the BigBlueBank system using the BANK transaction, as shown in Figure 7-1.

```
BigBlueBank        CUSTOMER MAINTENANCE SYSTEM        Date: 11-14-14
MENU  MAINMENU              MAIN MENU                 Time: 10:56:12

   OPTION    ACTION TO BE TAKEN


     1     Add a New Customer

     2     Browse an Existing Customer

     3     Modify Details about an existing Customer

     4     Browse Details about ALL the Customers

     5     Get a Credit Rating for an existing Customer

     X     Exit the Customer Maintenance System

           Select the Action to be performed


ENTER = Process the Option                          PF3 = Exit
```

Figure 7-1 Bank transaction

7.3 Migration checklist

For a detailed checklist about migrating from an older version to the current version of TXSeries for Multiplatforms, see the topic about Migrating to TXSeries for Multiplatforms V8.x from an earlier version under the Installing task in the TXSeries documentation library:

http://www.ibm.com/support/knowledgecenter/SSAL2T/welcome

7.4 Migration issues with BigBlueBank

This section describes the issues that we faced during the migration of the BigBlueBank system to the current version of TXSeries for Multiplatforms.

7.4.1 Unable to start the CICS region

Problem: The first time we started the region after the migration, the region failed to start. Example 7-15 shows the message as seen in the console.

Example 7-15 Console message

```
ERZ057001E/0154 11/16/05 15:43:29.284577159  1183802/0001      : Cannot
access message catalog cics.cat for message ERZ057002E
Please check if NLSPATH or LANG variable is set
ERZ057001E/0154 11/16/05 15:43:29.285688359  1183802/0001      : Cannot
access message catalog cics.cat for message ERZ057005E
Please check if NLSPATH or LANG variable is set
ERZ057001E/0094 11/16/05 15:43:29.286128679  1183802/0001      : Cannot
access message catalog cics.cat for message ERZ010003I
Please check if NLSPATH or LANG variable is set
ERZ057001E/0133 11/16/05 15:43:29.286738463  1183802/0001      : Cannot
access message catalog cics.cat for message ERZ010088E
Please check if NLSPATH or LANG variable is set
ERZ057001E/0175 11/16/05 15:43:29.287416857  1183802/0001      : Cannot
access message catalog cics.cat for message ERZ010043I
Please check if NLSPATH or LANG variable is set
```

From this message, it is clear that TXSeries is not able to read the language catalog files. We checked the `/etc/environment` file that is read by the AIX subsystem controller, and found that **`NLSPATH`** does not contain path names to TXSeries catalog.

Solution: We added the TXSeries language catalog to **`NLSPATH`** in the `/etc/environment`, as shown in Example 7-16.

Example 7-16 Adding the catalog to NLSPATH

```
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat:/usr/lpp/cics
/msg/%L/%N:/usr/lpp/cics/msg/C/%N:/usr/lpp/cicssm/lib/msg/en_US/%N
```

8

The BigBlueBank sample application

In this chapter, we describe the BigBlueBank sample application on which we have based our examples throughout this book. We describe the components of the application, and also provide instructions about how to install and run it.

This chapter provides information about the following topics:

8.1 The BigBlueBank application

BigBlueBank is an application designed as a technology demonstrator. It is not intended as an example of best practices in all aspects of programming. BigBlueBank has 3270-based panels and communication area (COMMAREA)-based programs. The programs have been written using various technologies and languages:

- Common Business Oriented Language (COBOL), in this case IBM COBOL
- C
- Web services
- IBM Customer Information Control System (IBM CICS) Foundation Classes
- Java class library for CICS (JCICS)

Figure 8-1 shows an overview of the BigBlueBank application.

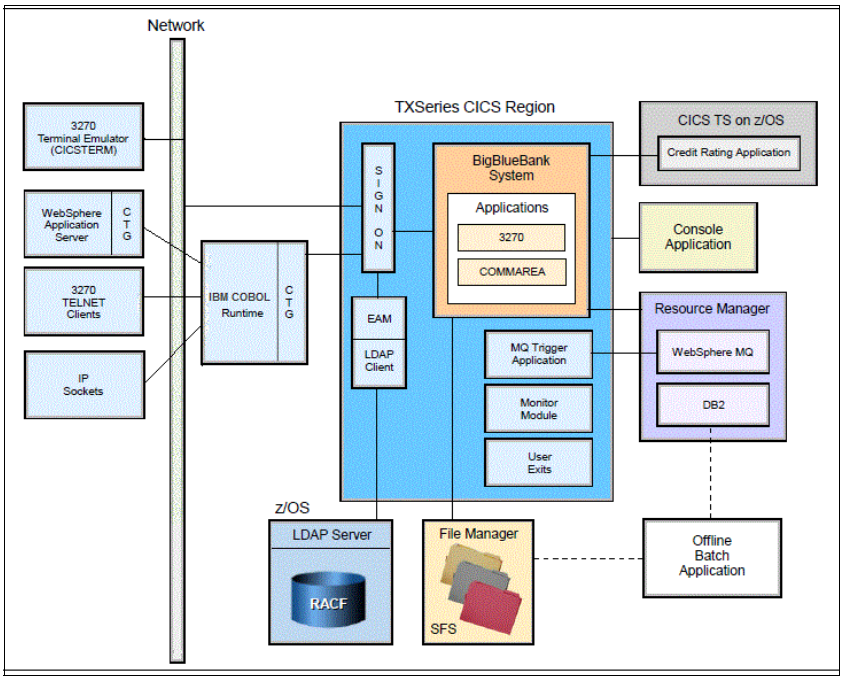


Figure 8-1 BigBlueBank application overview

8.1.1 A note on development language choice

We chose to develop this sample application in several different languages. The choice of a particular language for any given part of the system was made in a close-to-arbitrary manner. Frequently this was because the author of that particular part was most comfortable in that language.

We do not mean to imply that any language is best for any given part of the application, and (most importantly) we do not mean to imply that any given language *cannot* be used for any given part of the application.

The programs are installed by default into the following directories:

- `/ITSO/SA-Z036-R01` (on IBM AIX)
- `C:\ITSO\SA-Z036-R01` (on Microsoft Windows)

There are subdirectories (folders) within these folders, which are described in 8.3, “Server-side programs” on page 387, and 8.4, “Client-side programs” on page 391. For details about the installation and use of the BigBlueBank applications, see 3.1, “Preparation” on page 48, along with its following sections.

8.2 Installing the BigBlueBank sample application

In this section, we assume that IBM TXSeries for Multiplatforms is installed, and that the installation verification procedure (IVP) is already run to test basic functionality. We describe the basic installation of the BigBlueBank system. Other optional components are installed and configured in other sections of this book.

8.2.1 AIX

To install the sample application, complete the following steps:

1. Log in as root. Alternatively, you can add your user ID to the cics group and use it to login.
2. We provide the BigBlueBank system in a compressed file, `SG247185.zip`. Copy this to the `/tmp` directory, and create a directory into which this file is decompressed:

   ```
   mkdir -p /tmp/BigBlueBank
      cd /tmp/BigBlueBank
      unzip SG247185.zip
   ```

3. Create an environment variable to point to the installation directory:

   ```
   export BANK=/tmp/BigBlueBank/SG247185/demo
   ```

4. Change to the directory for the banking application region maintenance:

   ```
   cd $BANK/region
   ```

5. Run a script to create a new region, if required:

   ```
   ./reg_create
   ```

   When prompted, enter a name for the region to be created. You can choose to skip this step if you plan to use an existing region.

6. Set an environment variable to the name of the chosen region:

   ```
   export CICSREGION=<region_name>
   ```

7. Modify the region to enable the use of debugger for applications:

   ```
   cicsupdate -c rd -r $CICSREGION -P AllowDebugging=yes
   ```

   Important: All commands and options in TXSeries are case-sensitive.

8. Change to the `Table_Scripts` directory:

   ```
   cd $BANK/region/Table_Scripts
   ```

9. Run a script to create the following definitions:

   a. Transaction definitions (TD) entries:

      ```
      ./entry.tran
      ```

   b. Program definitions (PD) entries:

      ```
      ./entry.prog
      ```

   When prompted by each script, enter the region name.

10. Run a script to create the file definitions (FD) entries:

    ```
    ./entry.file
    ```

    When prompted, enter the region name, and the Structured File Server (SFS) name that identifies the SFS file system server.

    Remember: On some systems, there can be multiple SFS file servers. The machine name is the default server name.

11. Change to the banking directory to create data files:

    ```
    cd $BANK/files
    ```

12. Create the following files:

    a. The error messages file:

       ```
       ./create_BANKERR
       ```

b. The customer master file:

```
./create_BANKCUST
```

c. The account master file:

```
./create_BANKACCT
```

In each case, when prompted, enter the SFS name.

13. Initialize all of the banking application's data files with test data. This step is required for `ERRFILE` for the banking application to run. For the other files, the test data is optional. To load test data into the files:

    a. Change to the directory that contains the test data:

    ```
    cd $BANK/files/data
    ```

    b. Run a script:

    ```
    ./sdt
    ```

    c. When prompted, enter the SFS name.

    d. At the `cicssdt` command prompt, enter the following series of commands, which load the data from flat to sequential SFS files:

    i. **`ftos BANKERR`**

    When prompted for the data file, enter `BANKERR.sdt`.

    ii. **`ftos BANKCUST`**

    When prompted for the data file, enter `BANKCUST.sdt`.

    iii. **`ftos BANKACCT`**

    When prompted for the data file, enter `BANKACCT.sdt`.

    e. Exit from `cicssdt` with the **`quit`** command.

14. Change to the folder that contains basic mapping support (BMS) maps for the banking application:

    ```
    cd $BANK/app/BMS_Maps
    ```

15. Assemble all of the maps of the banking application using the following script:

    ```
    ./assemble_all
    ```

    When prompted, enter the region name to be used.

16. With the IBM COBOL version of BigBlueBank, certain functionalities exist, such as basic 3270, IBM CICS Transaction Gateway (CICS TG) aspects of the system, and connectivity to IBM WebSphere MQ and IBM DB2 on AIX:

    a. Change to the directory containing the IBM COBOL source for the banking application:

    ```
    cd $BANK/app/Sources
    ```

b. Compile all of the programs of the Banking application using the following script:

```
./compile_all
```

As before, when prompted, enter the region name.

17. Change to the banking region maintenance directory:

```
cd $BANK/region
```

18. Restart the newly created region:

```
./reg_restart
```

When prompted, enter the <*region_name*> to be used.

19. Verify the installation, and start a TXSeries CICS terminal session:

```
cicslterm -r $CICSREGION
```

20. Start the BigBlueBank application with the BANK transaction, as shown in Figure 8-2.

```
BigBlueBank              CUSTOMER MAINTENANCE SYSTEM         Date: 11-23-05
 MENU  MAINMENU                    MAIN MENU                  Time: 11:59:56


     OPTION    ACTION TO BE TAKEN



        1      Add a New Customer

        2      Browse an Existing Customer

        3      Modify Details about an existing Customer

        4      Browse Details about ALL the Customers

        5      Get a Credit Rating for an existing Customer

        X      Exit the Customer Maintenance System

               Select the Action to be performed


 ENTER = Process the Option                                        PF3 = Exit
```

Figure 8-2 Bank transaction

8.3 Server-side programs

We wrote the server-side programs in COBOL, C, C++, and Java. Table 8-1 lists the source code for the server application.

Table 8-1 Source code for the server application

Folder or file name	Contents
demo (demo\server on Windows)	
► app/BMS_Maps ► app/Sources ► app/Copybooks	Folders that contain the banking application source files
► /files ► /files/data	Contains files and folders that are used to create and initialize data files in the sample banking application
► /region ► /region/Table_Scripts	Contains files that are used to create and maintain regions, including the CICS resource definitions needed for the sample banking application
/app/BMS_Maps	
CADDSET.bms	Contains the Customer Add panel BMS map
CBROSET.bms	Contains the Customer Browse panel BMS map
CDEPSET.bms	Contains the Customer Deposit panel BMS map
CREDSET.bms	Contains the Customer Credit Check BMS map
CSBRSET.bms	Contains the Customer Start Browse panel BMS map
CUPDSET.bms	Contains the Customer Update panel BMS map
CXFRSET.bms	Contains the Customer Transfer panel BMS map
MENUSET.bms	Contains the Main Menu panel BMS map
assemble	Script that accepts the region name and BMS map name, and assembles that map
assemble_all	Script that accepts the region name and then assembles all of the BMS maps for the sample banking application
/app/Copybooks (COBOL copy members)	
ABENDWS.CPY	Working storage for the **ABEND** program
ACCMAST.CPY	File layout of the Account Master file
COMMAREA.CPY	Working storage for the COMMAREA-based programs

Folder or file name	Contents
CREDRATE.CPY	Working storage for the COMMAREA for the **CUSTCRED** program
CUSTADD.CPY	Working storage of the Customer Add panel
CUSTBROW.CPY	Working storage of the Customer Browse panel
CUSTCRED.CPY	Working storage of the Customer Credit Check panel
CUSTDPST.CPY	Working storage of the Customer Deposit panel
CUSTMAST.CPY	File layout of the Customer Master file
CUSTUPDT.CPY	Working storage of the Customer Update panel
CUSTXFER.CPY	Working storage of the Customer Transfer panel
ERRFILE.CPY	File layout of the Error Message file
FACDEFN.CPY	Working storage of the FAC definitions
HDRS-ENG.CPY	Working storage of the panel titles in English
MAINMENU.CPY	Working storage of the Main Menu panel
MQLINK.CPY	Linkage section for the call to **CUSTMQ**
SIGNON.CPY	Working storage of the SIGN ON panel
SQLLINK.CPY	Linkage section for the call to **CUSTSQL**
app/Sources	
CABEND.ccp	Program called whenever an abnormal end of task (abend) occurs
CDUMMY.ccp	Program called from **CABEND**, returns control to CICS
CUSTADD.ccp	Program to add new customer records to the Customer Master and Account Master files
CUSTBROW.ccp	Program to browse a single Customer record
CUSTCRED.ccp	Program to perform a customer credit check
CUSTDPST.ccp	Program to deposit and withdraw funds from customer accounts
CUSTECI.ccp	COMMAREA-based CICS Transaction Gateway program
CUSTLINK.ccp	Program called from the **CUSTBROW** program using the **EXEC CICS LINK** command for testing the **CDCN** utility
CUSTMQ.cbl	Program that gets called to write to a WebSphere MQ queue
CUSTSTBR.ccp	Program to browse all the customer records

Folder or file name	Contents
CUSTSQL.sqb	Program that gets called to write to a DB2 table
CUSTUPDT.ccp	Program to update the details about the customer
CUSTXFER.ccp	Program to transfer funds between different accounts of the customer
MAINMENU.ccp	Main menu program for the banking application
compile_all	Script that accepts the region name and then compiles all of the IBM COBOL programs for the banking application
z/OS (Mainframe COBOL source and copy files)	
CREDAP.ccp	COMMAREA COBOL program to return a Credit Score
COMMAREA	Working storage file with the COMMAREA specification
ABENDWS	Working storage file with items used in case of an abend
JCICS (Java program to return credit ratings)	
README.txt	A readme file describing the files we supply
JCICS/BANKCRED/com/ibm/bank/bankcred	
BankCredApp.java	The main **BANKCRED** application
DplLink.java	Distributed program link (DPL) implementation file
LogMsg.java	Temporary storage queue (TSQ) logging implementation file
VsamKsds.java	Virtual Storage Access Method (VSAM) key-sequenced data set (KSDS) access implementation file
CICS Foundation Classes (C++ program to return credit ratings)	
README.txt	A readme file describing the files we supply
Makefile	A makefile to build the application
bankcred.cpp	The main BankCred application to get the credit rating
bankcred.hpp	Header file with BankCred and COMMAREA class definitions
C\MSVC.NET Project\BigBlueBank (Microsoft Visual Studio .NET project for server-side programs)	
► BigBlueBank.sdf ► BigBlueBank.sln ► BigBlueBank.vcxproj	Visual Studio .NET project files
CustBrowse.ccs	Program to browse a single customer record

Folder or file name	Contents
`custeci.ccs`	COMMAREA-based CICS Transaction Gateway program
`CUSTECI.h`	A header file for the CICS Transaction Gateway programs
`CustGet.ccs`	A program to get a single customer record
`CustSignOn.ccs`	A program to perform a CICS VERIFY PASSWORD
`CustUpdate.ccs`	A program to update a customer record
`CustUtility.c`	A utility program
External authentication manager (EAM) (C EAM module)	
`README.txt`	A readme file describing the files we supply
`Makefile`	The makefile to build the EAM module
`ldap_search.c`	Implementation file to search profiles in IBM z/OS Resource Access Control Facility (IBM RACF) repository
`ldap_modify.c`	Implementation file for modifying user attributes in RACF
`ldap_sample.h`	Supporting header file
`cics_eam_ldap.c`	Main implementation file
`/MQTrigger`	
`cmqtrig.ccs`	WebSphere MQ Trigger monitor
`accchgal.ccs`	WebSphere MQ Trigger application
`/Monitor`	
`monitor.ccs`	CICS monitor and statistics program
`/Control`	
`startprg.ccs`	CICS region startup program
`purge.ccs`	CICS region shutdown program

8.4 Client-side programs

We wrote the client-side external call interface (ECI) programs in COBOL and Java. The server-side programs can also be accessed through the 3270 screen-based terminal emulators.

Table 8-2 lists the source code for the client application.

Table 8-2 Source code for the client application

Folder or file name	Contents
demo\Client	
► Client ► Client\Copylib ► Client\Source ► Client\Object ► Client\Resource	Folders that contain the client application source files (other folders are present). These are used only as *Placeholders* for AcuBench. No files are created within them.
cblconfi	Runtime configuration file for use with the client application.
CTGDemo.psf	Program Structure File used by AcuBench.
Project.pjt Project.wif	Workspace and project files used by AcuBench.
\Client\Copylib	
ECI_CommArea.cpy	Working storage for the COMMAREA.
\Client\Object	
AcuBenchPrint.Dll	Dynamic link library (DLL) provided by AcuBench for Windows Printing (not required).
CTGDemo.acu	The application object code.
\Client\Resource	
buttons.bmp	Bitmap containing the images for the application buttons.
\Client\Source	
CTGDemo.cbl	Source code for the client application.

Tip: If you compile the client-side programs from the command line, you have to use the following options:

- **`-Ce=CBL`**: The CICS ECI copy members have an extension of CBL, which is not coded in the COPY statement. This option tells the compiler to add it.
- **`-Da4`**: Sets the data alignment modulus for 01- and 77-level items to 4 bytes.
- **`-Dw32`**: Sets the maximum alignment word size to 32 bits.
- **`-Dz`**: Computes size errors by examining the actual storage of the item. This is needed to ensure that the CICS return codes are correctly stored.
- **`-Ga`**: Specifies that debugging symbols must be included in the object code.

Set the `COPYPATH` to point to the location of the required copybooks. Then compile the client-side program.

9

Application development and administration choices

This chapter highlights some of the techniques that help implement effective application programs, eliminate common errors, ease long-term maintenance, and increase the chances for solution extensibility in a middleware environment.

This chapter provides information about the following topics:

9.1 Application development considerations

Application development in any of the middleware environments is an important task that has to be considered carefully. Because the execution of the application is controlled by the transaction coordinator, there are always certain considerations on what an application can or cannot do, to maintain the integrity of the entire transactional system. This section describes some of the important application development-related considerations.

9.2 Application design considerations

Application design is the first step of an application development. The following sections highlight some of the choices in this stage.

9.2.1 Dividing applications into components

An important principle of business application design is to separate program code into *components*. Components are not the same as **`objects`**, nor are they the divide and conquer aspect of implementing a large project.

Components are about managing a large complex information technology (IT) environment and reuse. With good component design, you can enhance and change a business application quickly and easily. Figure 9-1 shows the components of an application.

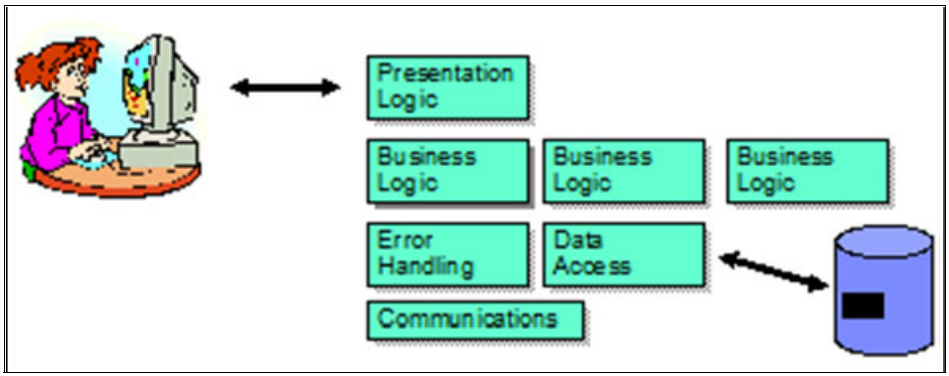


Figure 9-1 Components of an application

The following list describes some general guidelines for developing components:

- Different responsibilities within the overall application. One program deals with accounts, another with cash, and so on.
- Different types of responsibility:
  - One program handles presentation (presentation logic).
  - Another program performs interest rate calculations (business logic).
- Different hardware and software platforms. Use custom modules to tailor applications to your environment, enabling rapid redeployment to new environments.

9.2.2 Error handling within applications

Design error handling into the application. Remember the following phrases:

- “Anything that can happen will happen.”
- “Absolutely anything can happen.”

For example, you can categorize errors into the following types. Decide in advance how to handle each type of error:

- Conditions that are not normal from the IBM Customer Information Control System (IBM CICS) point of view, but are anticipated in the program
- Conditions caused by user errors and input data errors
- Conditions caused by omissions or errors in application code
- Errors caused by mismatches between applications and CICS resources, for example, file not found
- Errors relating to hardware or other systems conditions beyond the application program control

9.2.3 Separating program components

Figure 9-2 shows the logical levels of the program components.

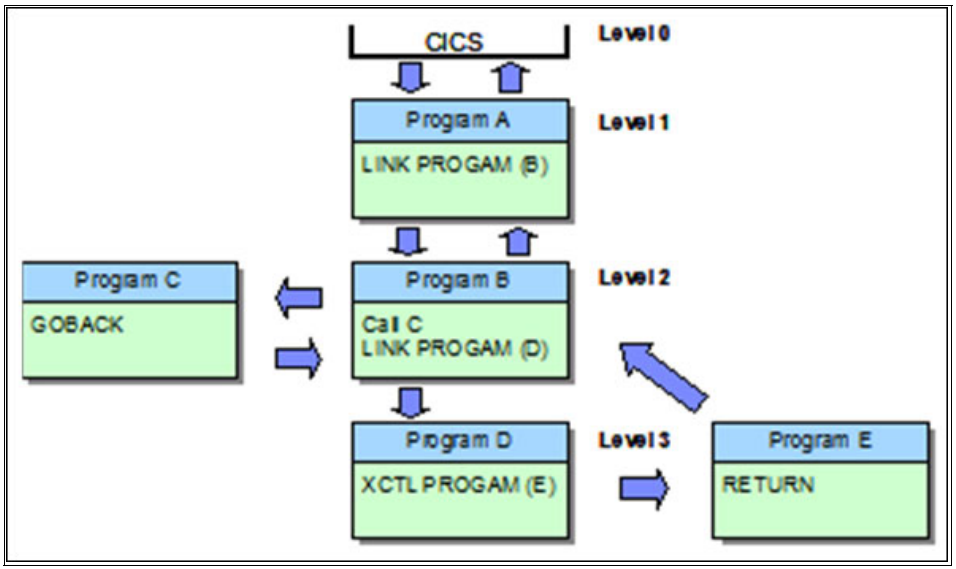


Figure 9-2 Separating program components: Logical levels

Use the CICS API commands **LINK** and **XCTL** to call or transfer to other CICS programs or modules. Use this technique to help split an application into components. You can also use native language calls (Common Business Oriented Language (COBOL) **CALL** or C functions) to call programs or modules that have common code or libraries, but no CICS code (for example, program C).

In COBOL, you can use the **CALL** statement to run another COBOL program as part of the calling one. The called program can either be statically linked-in with the caller, or it can be dynamically loaded. Linking in the called program statically is faster but inflexible. Loading a called program dynamically can be slower, but it is more flexible. Furthermore, by dynamically loading a called program, you can arrange for the called program to be shared.

In this instance, the first call to it is comparatively slow, but subsequent calls are faster while it remains loaded. You should be aware that when you take this approach, you are *not* descending a CICS logical level.

The C language encourages the division of related functions into one or more source files that are compiled separately and then combined to form an executable object. Each of the source files that you combine into a single CICS program can include **EXEC CICS** statements.

However, again note that when you make a function call in C to a function that includes the CICS application programming interface (API), you are *not* descending a CICS logical level. Logical levels only apply to the use of the **LINK** command. If you have to inherit the same set of attention identifier handlers as the calling function, you must make a function call.

If your called function requires a new set, you have to make it into a CICS program and call it with **LINK** or **XCTL**. It is more efficient to call it as a C function. C programs can include static data (similar to working storage in COBOL). CICS programs written in C are given a fresh copy of their static data for only the first instance of the program.

9.2.4 Reusing business logic

Figure 9-3 shows the components (from Figure 9-1 on page 394) that are reused without change, including the business logic, error handling, data access, and presentation logic.

The socket client, C external call interface (ECI) client, socket server, and Java ECI client components are the new components that must be provided for a particular client environment. By reusing well-designed business logic, an application can be opened up to different client channels, with minimum effort and no rework of the existing server components.

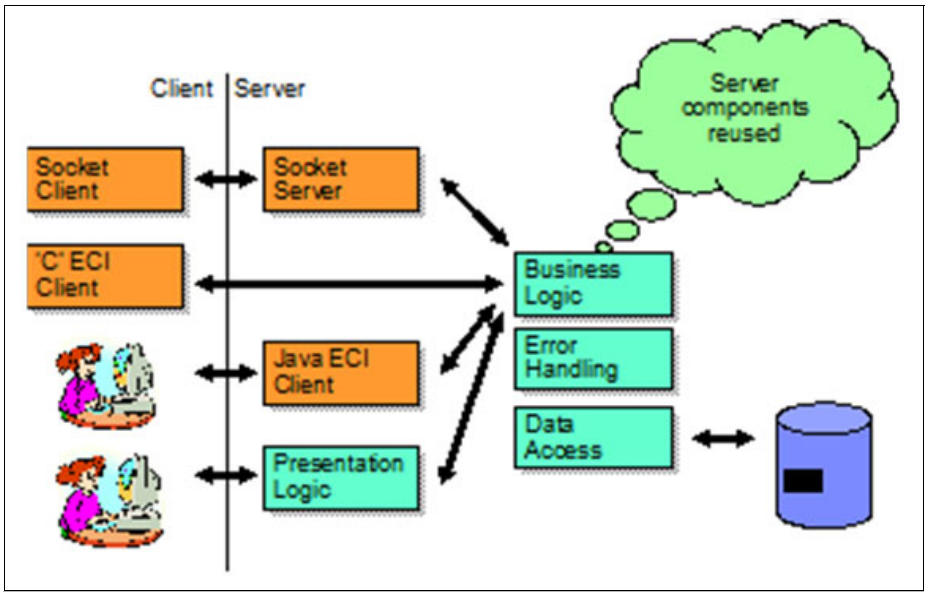


Figure 9-3 Reusing business logic

9.2.5 Avoiding conversational programs

Conversational programs (Figure 9-4) increase the length of logical unit of work (LUW). This locks the resources for the duration of the LUW.

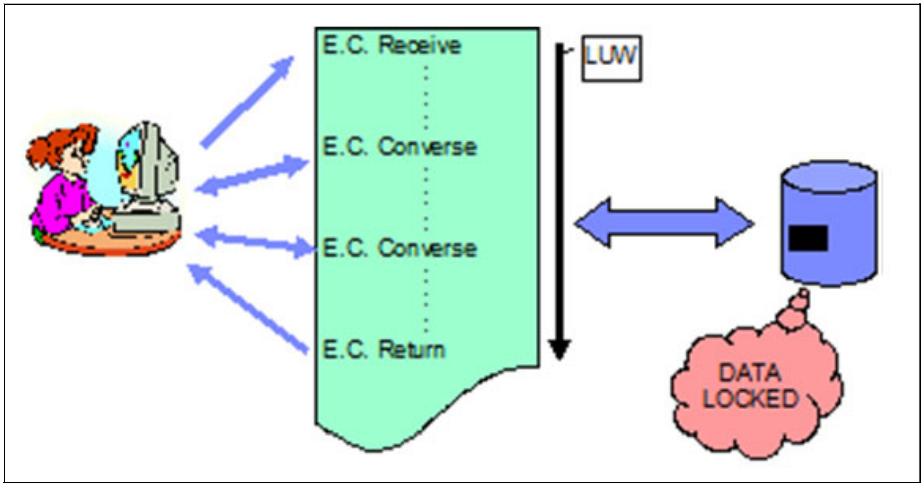


Figure 9-4 Conversational programs

Figure 9-5 shows non-conversational program samples.

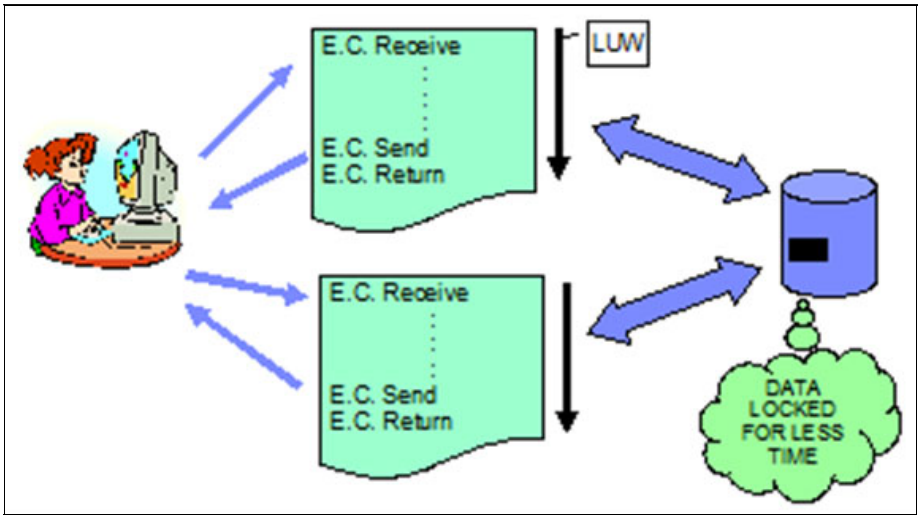


Figure 9-5 Non-conversational programs

To avoid conversational transactions, it is better to have one of the following options:

- Non-conversational program. There is only one request and reply.
- Pseudo-conversational program. There are several requests and replies.

With the first approach, you have to handle the following problems:

- Maintaining the state between requests:
  - If a terminal-based client uses the terminal
  - If a non-terminal-based client uses an application-controlled state, for example, communication area (COMMAREA)
- Affinity. Ensure that one of the following states is true:
  - The next request in the conversation ends up in the same CICS region.
  - Another CICS region can access the state saved by the first.

To avoid these problems, CICS has the following features to store the state:

- Transaction work area (TWA)
- COMMAREA
- Terminal control table user area (TCTUA)
- Common work area (CWA)
- Temporary storage (TS) or transient data queues (TDQs)
- Files
- Display panel
- Shared storage

9.2.6 Choosing the correct data storage

There are two types of storage available, *TASK PRIVATE* and *TASK SHARED*. Using the wrong type of storage affects performance and wastes storage.

Task-private storage

Task-private storage is private to the task and cannot be addressed by any other CICS task in the system. CICS releases all of the task-private storage that is associated with a task when the task is ended normally or undergoes an abnormal end of task (abend). Task-private data is typically stored in the following categories of private storages:

- Transaction work area

  The size of the transaction work area is determined by the `TWASize` option in the transaction definitions (TD). If this is given a nonzero value, the TWA is always allocated, it lasts for the entire duration of the transaction, and it is accessible to all of the programs in the transaction. The processor usage associated with using the TWA is minimal. You do not require a **GETMAIN** command to access it. You can address it using a single **ADDRESS** command.

The TWA is suitable for fairly small data storage requirements, and for larger requirements that are both relatively fixed in size and are used more or less for the duration of the transaction. Because the TWA exists for the entire transaction, a large TWA size has greater benefit for conversational transactions than for pseudo-conversational transactions. It is only available to local programs, not to remote ones.

- User storage acquired by **GETMAIN** without the **SHARED** option

  GETMAIN storage is allocated with a mechanism similar to standard library calls, such as **malloc** and **free** routines. It comes from the process data segment. CICS imposes a limit, configured in the region definitions (RD), on the amount of this type of storage that is given to a transaction program. This class of storage is reclaimed by CICS if the transaction program experiences an abend, or ends without releasing it.

  User storage is available to all the programs in a transaction, but some effort is required to pass it across a **LINK** or **XCTL**. The storage can be obtained just when the transaction requires it (using the **GETMAIN** command), and released as soon as it is no longer required (using the **FREEMAIN** command). Consequently, user storage is useful for large storage requirements that are either variable in size, or have a duration that is shorter than the transaction.

  A **GETMAIN** command involves a large amount of processor usage. Therefore, use it only for large amounts of storage. For smaller storage amounts, use the TWA, or group the requests together into a larger request. Although the storage acquired by a **GETMAIN** command can be held somewhat longer when using combined requests, the processor usage and the reference set size are both reduced.

- Communication area

  The COMMAREA option of the **LINK** and **XCTL** commands specifies the name of a data area (known as a communication area) in which data is passed to the program that is called. In a similar manner, the **COMMAREA** option of the **RETURN** command specifies the name of a communication area in which data is passed to the transaction identified in the **TRANSID** option. A maximum of 32 kilobytes (KB) of data can be passed in the COMMAREA.

- INPUTMSG option

  The **INPUTMSG** option of the **LINK**, **XCTL**, and **RETURN** commands provides another way of specifying the name of a data area that is to be passed to the program that is called. In this case, the called program obtains the data by processing a **RECEIVE** command. This option enables you to start application programs to obtain initial terminal input. These programs are written to be started directly from a terminal, and contain **RECEIVE** commands.

- Program storage

  In COBOL, when the **LINK** and **XCTL** commands start the program, a new copy of the working storage is loaded. However, programs called with the **CALL** statement use existing working storage if the program has already been called.

  In C and C++, recursive invocations of a program in the same transaction share one copy of its static and external data. You can use the C language function **alloca** to allocate stack storage for automatic variables and dynamically sized automatic variables.

  This storage is the quickest to allocate and deallocate, and is automatically cleaned up when the function exits or the transaction program abend occurs. The application programmer must ensure that this memory is freed at an appropriate time. Because it is not simple to defend against the possibility that a transaction program can experience an abend while holding malloc storage, it is normal for transaction programs to use the **GETMAIN** command instead.

Task-shared storage

Task-shared storage is shared between all CICS tasks. It can be accessed by multiple tasks. All of the synchronization to these areas is the responsibility of the applications that want to use them. Users can choose the following storage to share data between tasks:

- Temporary storage

  TS is the primary CICS facility for storing data that must be available to multiple transactions. Data items in TS are kept in queues whose names are assigned dynamically by the program storing the data. Unlike TDQs, you do not have to define temporary storage queues (TSQs) in the RDs.

  A TSQ containing multiple items can be seen as a miniature file whose records can be addressed either sequentially, or directly by item number. If a queue contains only a single item, it can be seen as a named scratch pad area.

  TS is implemented in two different ways. Which one is used for a particular queue is determined by what is specified on the command that creates the first item. **MAIN** means that the queue is kept in memory, and **AUXILIARY** means that the queue is written to disk storage. For each method, CICS maintains an index of items in the main storage. Both of these methods have certain characteristics that you should note:

  – The main TS requires much more virtual storage than the auxiliary TS does. In general, you should use it only for small queues that have short lifetimes, or are accessed frequently. The auxiliary TS is specifically designed for relatively large amounts of data that have a relatively long lifetime, or that are accessed infrequently.

- You can make queues in the auxiliary storage recoverable, but not queues in the main storage. Only one transaction at a time can update a recoverable TSQ. If you choose to make queues recoverable, note that there is a possibility of enqueues.
- If a task tries to write to the TS and there is no space available, CICS suspends it. The task is not resumed until some other task frees the necessary space in the file. This can produce unexplained response delays, especially if the waiting task owns exclusive-use resources, in which case all other tasks that require those resources must also wait. You can use the **NOSUSPEND** option to avoid this.
- It can be more efficient to use the main TS exclusively in low-volume systems that have no need for recovery.
- You must use a CICS command every time data is written to or read from a TSQ, and CICS must find or insert the data using its internal index. This means that the resource cost for using the main TS is greater than for the CWA or terminal user area. With auxiliary storage, which is often the most frequently used, there is usually file input/output (I/O) as well, which increases resource use even more.
- You do not have to allocate TS until it is required. You have to keep it only while it is required, and the item size is not fixed until you issue the command that creates it. This makes it a good choice for relatively high-volume data, and data that varies in length or duration.
- The fact that TSQs are named as they are created provides a powerful form of random access to saved data. You can access scratch pad areas for terminals, file records, and so on, just by including the terminal name or record key in the queue name.
- Resource-level protection is available for temporary storage.

► Transient data

You can use transient data for many of the purposes for which you use auxiliary temporary storage, but there are some important differences:

- TDQ names must be defined in the transient data definitions (TDD) before CICS is started. You cannot define them arbitrarily at the time when the data is created. Therefore, transient data does not have the same dynamic characteristics as TS.
- TDQs must be read sequentially, and each item can be read only once. That is, after a transaction reads an item, that item is removed from the queue. It is not available to any other transaction. In contrast, items in the TSQs can be read either sequentially or directly (by item number). They can be read any number of times, and are never removed from the queue until the entire queue is purged.

These two characteristics make transient data inappropriate for scratch pad data, but suitable for queued data, such as audit trails and output to be printed. In fact, for data that is read sequentially once, transient data is preferable to TS.

- Items in a TSQ can be changed. However, items in transient data cannot be changed.
- TDQs are always written to a data set. (There is no form of transient data that corresponds to main TS.)
- You can define TDQs so that writing items to the queue causes a specific transaction to be initiated (for example, to process the queue). TS does not have anything that corresponds to this trigger mechanism, although you might be able to use a **START** command to perform a similar function.
- Transient data has more varied recovery options than TS. It can be physically or logically recoverable.
- Because the commands for intrapartition and extrapartition transient data are identical, you can switch easily between the internal CICS facility (intrapartition) and an external data set. To do this, you only have to change the TDD, not your application programs. TS has no corresponding function of this kind.

► The common work area

CWA is a single control block that is allocated at system startup time, and it exists for the entire region lifetime. The size is fixed by specifying a value for the **CWASize** parameter in the RD. The CWA has the following characteristics:

- There is almost no resource used in storing or retrieving data from the CWA. Transaction programs must issue one **ADDRESS** command to get the address of the area but, after that, they can access it directly.
- Data in the CWA is not recovered if a transaction or the system fails.
- It is not subject to resource-level security.
- CICS does not regulate the use of the CWA. All programs in all applications that use the CWA must follow the same rules for shared use. The content of and access to the CWA is one of the design issues that you have to describe with application developers. It is sensible to supply a copybook or include a file to describe the contents.
- The CWA is especially suitable for small amounts of data, such as status information, that is read or updated frequently by multiple programs in an application.
- The CWA is not suitable for large-volume data or short-lived data, because it is always allocated.

- You must ensure that data used in one transaction does not overlay data used in another, by following the rules for shared use discussed previously.
- You must ensure that programs do not overrun the end of the CWA, because this corrupts the storage areas in the task shared pool, causing other transactions to fail.

► Terminal user area

The TCTUA is defined using the **TCTUALen** option in the terminal definitions (WD). If this length is not zero, the address of the area can be obtained with the **ADDRESS** command. Terminal user areas have the following characteristics in common with the CWA:

- Minimal processor usage (only one command required)
- No recovery
- No resource-level security
- No regulation of use by CICS
- Fixed length
- Unsuitable for large-volume data or short-lived data

However, unlike the CWA, the terminal user area for a particular terminal is usually shared only among transactions using that terminal. Therefore, it is useful for storing small amounts of data of fairly standard length between a series of pseudo-conversational transactions. Another difference is that it is not necessarily permanently allocated, because the terminal user area only exists while the WD is set up.

The TCTUA is allocated from region startup for non-autoinstall terminals and is allocated for autoinstall terminals when the WD is added. The allocated region is discarded for autoinstalled terminals when their destination is deleted.

Using the terminal user area in this way does not require special discipline among using transactions, because data is always read by the transaction following the one that wrote it. However, if you use terminal user areas to store longer-term data (for example, terminal or operator information required by an entire application), they require the same care as the CWA to ensure that data used in one transaction does not overlay data used in another.

You should also take care not to exceed the length of the allocated terminal user areas, because this causes corruption in other storage areas allocated from the CICS task shared pool.

► Display panel

You can also store data between pseudo-conversational transactions from a display terminal in the panel itself. For example, if users make errors in data that they enter, the transaction usually points out the errors on the panel (with highlights or messages), sets the next transaction identifier to point to itself (so that it processes the corrected input), and returns to CICS.

Saving the data on the panel is easy to code, but has two limitations:

- You should not use it with panels that contain large amounts of data if the likelihood of errors is high. This is because of the additional line traffic needed to resend the unchanged data. This does not apply to locally attached terminals.
- When the user presses the CLEAR key, the panel data is lost, and the transaction must be able to recover from this. You can avoid this by defining the CLEAR key to mean **CANCEL** or **QUIT**, if this is appropriate for the application concerned.

- Operating system files

  You can also use OS files to save data between transactions. This method probably has the largest resource requirements in terms of instructions processed, buffers, control blocks, and user programming requirements, but does provide extra function and flexibility. You can define files as recoverable resources, and also log changes to them for forward recovery. You can also use resource-level security.

- **GETMAIN** with **SHARED**

  GETMAIN SHARED is allocated from storage shared between all CICS tasks. The total available is configured in the region database. **GETMAIN SHARED** remains allocated until explicitly freed by a transaction program. When you issue a **GETMAIN SHARED** command, you can pass the address of that area of storage to another transaction in the following ways:

  - Using the CWA
  - Using a shared CICS area
  - Using a shared operating system (OS) area
  - Using the COMMAREA

  Important: It is the responsibility of application programmers to monitor storage area addresses allocated by **GETMAIN SHARED**. Unlike ordinary **GETMAIN** storage, there is no implicit means of freeing it. Such storage areas should be released by a call to **FREEMAIN** by some application program (in a set of cooperating tasks). Otherwise, the shared storage is not freed until region termination.

9.3 Application programming considerations

Every middleware imposes certain considerations when implementing the applications to work in its environment, such as coding habits. This is primarily necessary to maintain the integrity of the transactional system. The following sections introduce some of the considerations on TXSeries.

9.3.1 RETURN IMMEDIATE

When you link to a program using the **LINK** command, use the **RETURN IMMEDIATE** option to prevent automatic transaction initiations (ATIs) hijacking the terminal.

9.3.2 SYNCPOINT on RETURN

If possible, use **SYNCONRETURN** to minimize the size of LUW.

9.3.3 Implementing COBOL CALL or C routines

Use COBOL **CALL** or C functions for modules with no CICS commands, because these are faster.

9.3.4 NULL fill communication areas before use

Always prefill the COMMAREA with nulls `x'00'` before using fill from byte 0. CICS optimizes the data transfer by stripping trailing nulls before sending, and then restores before handing to the receiving program.

This speeds up performance in passing data from one program to another, especially when the data has to be sent to the machines or the network. This applies to communication areas sent and returned (that is, both ways on the distributed program link (DPL)). It applies regardless of the total communication area size.

9.3.5 Keeping logical unit of work short

Atomicity, consistency, isolation, and durability (ACID) transactions should be short. Short is a relative term. In a large system with hundreds of transactions per second, short is the order of milliseconds, definitely subsecond. It is up to the designer to decide how long a transaction should last.

However, in almost all cases, it should not include user think time. This means that it must not ask for user input in the middle of a transaction. The objective is to make an LUW as short as possible. The longer the LUW, the longer the recoverable or updated resources are locked.

9.3.6 Updating resources close to a sync point

Consider updating a resource close to a sync point (if possible). This minimizes the duration of lock on the resource that is being updated.

9.3.7 Minimizing enquiry character times

When using enqueue (ENQ) to protect a resource, the time between the ENQ and de-enqueue (DEQ) should be as short as possible.

9.3.8 Minimizing terminal traffic

The **MAPONLY** option sends only the constant data in a map, and does not merge any variable data from the program. The resulting data stream is not always shorter, but the operation has a shorter path length in basic mapping support (BMS). When you send a skeleton panel to be used for data entry, you can often use **MAPONLY**.

Sending only changed fields is important when, for example, a message is added to the panel, or one or two fields on an input panel are highlighted to show errors. In these situations, you must use the **DATAONLY** option to send a map that consists of nulls except for the changed fields.

For fields where only the attribute byte has changed, you have to send only that byte, and send the remaining fields as nulls. BMS uses this input to build a data stream consisting of only the fields in question, and all other fields on the panel remain unchanged.

The modified data tag (MDT) for a field is usually turned on by the 3270 hardware when the user enters data into a field. However, you can also turn on the tag when you send a map to the panel, either by specifying **FSET** in the map or by sending an override attribute byte that has the tag on. You should never set the tag on in this way for a field that is constant in the map, or for a field that has no label (and is not sent to the program that receives the map).

Also, you do not usually have to specify **FSET** for an ordinary input field. This is because the MDT is turned on automatically in any field in which the user enters data. This is then included in the next **RECEIVE** command. These tags remain on, no matter how many times the panel is sent, until explicitly turned off by the program (by the **FRSET**, **ERASEAUP**, or **ERASE** option, or by an override attribute with the tag off).

FRSET turns off the MDTs, so that fields entered before that write are not present unless the user re-enters them the next time. If you are dealing with a relatively full panel and a process where there might be several error cycles (or repeat transmissions for some other reason), this can be a substantial saving.

However, because only changed fields are sent on subsequent reads, the program must save input from each cycle, and merge the new data with the old. This is not necessary if you are not using **FRSET**, because the MDTs remain on, and all fields are sent regardless of when they were entered.

Sending fields to the panel that consist entirely of blanks, or that are completed on the right by trailing blanks, wastes line capacity. The only case where BMS requires you to do this is when you have to erase a field on the panel that currently contains data, or to replace it with data shorter than that currently on the panel, without changing the rest of the panel.

This is because, when BMS builds the data stream representing your map, it includes blanks (`X'40'`), but omits nulls (`X'00'`). This makes the output data stream shorter.

BMS omits any field whose first data character is null, regardless of subsequent characters in the field. It requires you to initialize to nulls any area to be used to build a map. This is done by moving nulls (`X'00'`) to the `mapnameO` field in the symbolic map structure.

BMS uses nulls in attribute positions and in the first position of data to indicate that no change is to be made to the value in the map. If you are reusing a map area in a program or in a terminal input/output area (TIOA), you should take special care to clear it in this way.

Often, users are required to complete the same panel several times. Only the data changes on each cycle, but the titles, field labels, instructions, and so on remain unchanged. In this situation, when an entry is accepted and processed, you can respond with a **SEND CONTROL ERASEAUP** command (or a map that contains only a short confirmation message and specifies the **ERASEAUP** option).

This causes all of the unprotected fields on the panel (that is, all of the input data from the last entry) to be erased and to have their MDTs reset. The labels and other text, which are in protected fields, are unchanged. The panel is ready for the next data-entry cycle, and only the necessary data has been sent.

Design the application so that all of the entry fields are processed in a single pass. All the errors are then marked on the panel. This is more efficient than ending the validation and responding back to the panel for every problem.

When you send unformatted data to the panel, or create a formatted panel outside BMS, you can compress the data further by inserting set buffer address (SBA) and repeat to address (RA) orders into the data stream. SBA enables you to position data on the panel, and RA causes the character following it to be generated from the current point in the buffer until a specified ending address.

SBA is useful whenever there are substantial unused areas on the panel that are followed by data. RA is useful when there are long sequences of the same character, such as blanks or dashes, on the panel. However, note that the speed with which RA processes is not uniform across all models of 3270 control units. You must verify how it applies to your configuration before use.

Use the **CONVERSE** command rather than the **SEND** and **RECEIVE** commands (or a **SEND**, **WAIT**, and **RECEIVE** command sequence if your program is conversational). They are functionally equivalent, but the **CONVERSE** command crosses the CICS services interface only once, which saves processor time.

9.3.9 Checking for errors

Always code **RESP**, **RESP2**, or both, and use the **DFHRESP** function to test **RESP** and **RESP2** values. This forces CICS to hand control back to the program so that it can handle errors, and avoids CICS default handling of errors, which can be to stop the program.

9.3.10 Avoiding deadlock

Deadlock is a well-known name in the computer field. It always affects the system seriously.

How deadlock happens

Consider a case where there are two resources being worked by two different tasks. There can be a scenario where the first task locks the first resource and waits for the second resource, while the second task has locked the second resource and waits for the first resource. This results in a deadlock. At this stage, the only option available is to stop one of the tasks, rolling back any updates and let the other task complete normally.

If both resources are CICS resources, the task whose deadlock timeout period elapses first is abnormally ended, and its CICS resources are released. (It is possible for both tasks to time out simultaneously.) If both these tasks do not have a `DeadLockTimeout` attribute value in the TD, they remain suspended indefinitely, unless one of them is abnormally ended. CICS then backs out the abnormally ended task.

Figure 9-6 illustrates a deadlock scenario.

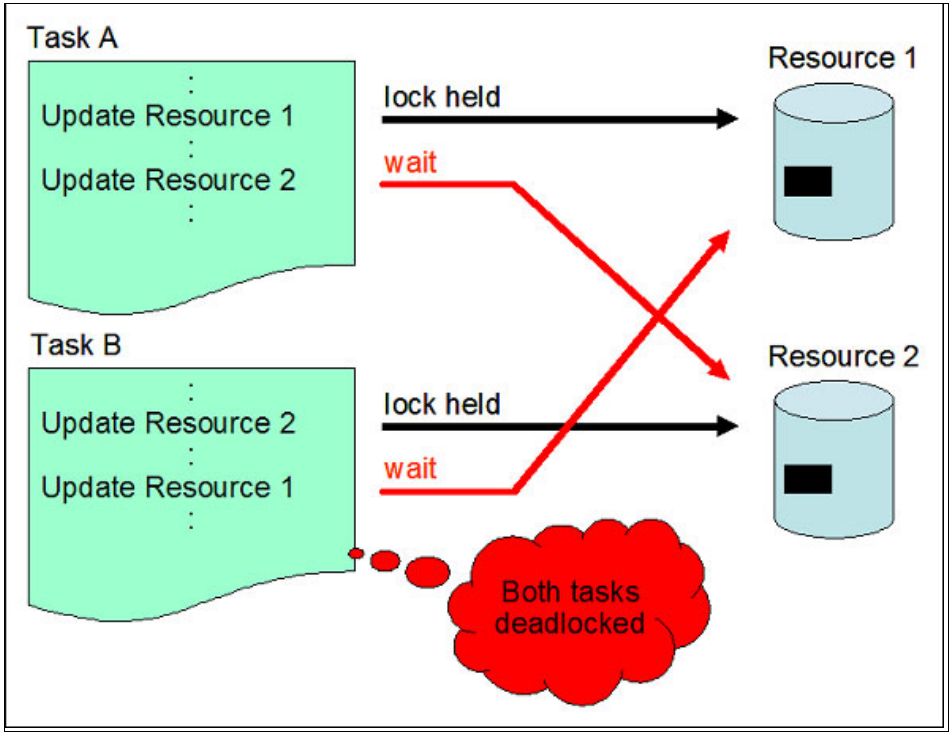


Figure 9-6 Deadlock scenario

Avoiding deadlock passively

The default action for the `ENQBUSY`, `NOJBUFSP`, `NOSPACE`, `QBUSY`, and `SYSBUSY` conditions is to suspend the execution of the application until the required resource (for example, a queue) becomes available. Then, the action resumes processing the command. The following commands can cause these conditions:

- **`ALLOCATE`**
- **`CONNECT PROCESS`**
- **`ENQ`**
- **`JOURNAL`**
- **`READQ TD`**
- **`WRITEQ TS`**

With these commands, you can use the **`NOSUSPEND`** option to inhibit this waiting, and to cause an immediate return to the instruction in the application program following the command. (In the case of the **`ALLOCATE`** command, this option is known as the **`NOQUEUE`** option.)

If you do not use the **NOSUSPEND** option, the suspended applications periodically attempt to obtain the required resource until it becomes available. This can use significant resources. For example, processor time can be used, or a terminal lock can prevent the user from typing anything until the resource is available.

Avoiding deadlock actively

You can avoid deadlocks by following these rules:

- All applications that update (modify) multiple resources must do so *in the same order*. For instance, if a transaction is updating more than one record in a data set, it can do so in an ascending key order. A transaction that is accessing more than one file must always do so in the same predefined sequence of files.

  If a data set has an alternative index, beware of mixing transactions that perform several updates by the base key with transactions that perform several updates by the alternative key. Assume that the transactions that perform updates always access records in the ascending key sequence. Then transactions that perform all updates by the base key do not deadlock with other transactions that perform all updates by the base key.

  Similarly, transactions that perform all updates by the alternative key do not deadlock with other transactions that perform all updates by the alternative key. However, transactions that perform all updates by the base key might deadlock with transactions that perform all updates by the alternative key.

  This is because the key that is locked is always the base key. Therefore, a transaction performing updates by the alternative key might be acquiring locks in a different order than a transaction performing updates by the base key.

- Explicit enqueuing conventions should also be the subject of your site development standards so that all applications meet the following standards:
  - Enqueue using the same character string.
  - Use those strings in the same sequence.

- An application that issues a **READ UPDATE** command must follow it with a **REWRITE**, **DELETE** without **RIDFLD**, or **UNLOCK** command to release the position before doing anything else to the file. Alternatively, it should include the **TOKEN** option with both parts of each update request.

- An application must end all browses on a file by using **ENDBR** commands (therefore releasing the Virtual Storage Access Method (VSAM) lock) before issuing a **READ UPDATE**, **WRITE**, or **DELETE** with **RIDFLD** command to the file.

9.3.11 Avoiding CICS reserved letters and names

Avoid using variable names, such as `EXEC`, `CICS`, and `END-EXEC`. Avoid abend names starting with the letter “A” or “U”.

9.3.12 Avoiding writing to CICS log files (CSMT/console)

`WRITE OPERATOR` writes messages to the CICS System Termination (CSMT) TD queue (the `CSMT.out` file). This means that CICS and application messages are mixed together. This sometimes causes an inconvenience in analyzing problems.

Similarly, the usage of the **`STDOUT`** stream in the application writes to the CICS `console.msg` file. The usage of the **`STDERR`** stream in the application writes to the `console.`*<nnnnnn>* file. For example, avoid using **`DISPLAY`** statements in COBOL. Instead, you can turn on certain compiler options that might implicitly redirect **`DISPLAY`** messages to a separate OS file than to the **`STDOUT`** stream by default.

We suggest that you write application-specific messages to a separate TD queue. It is also useful to log the time stamp and the application-specific messages. This helps in correlating with the other logs if there is a failure.

9.3.13 Thread safety

Compile all CICS programs by using the thread-safe version of a runtime language library. The applications are run under the CICS application server process, which is a multi-threaded process. Certain types of functions might not behave correctly when called consecutively from two or more threads.

Do not use these functions, or call them using only a serialization technique. For example, do not use any function that keeps static data between calls (such as a C library call, `ctime`):

- On CICS on open systems

  The operating system provides a replacement library (as an example `libc_r` for `libc`), which contains thread-safe replacements for the functions that are not thread safe. In some cases, function names have been changed. For example, the replacement thread-safe version of `ctime` is called `ctime_r`.

- On CICS for Windows

  Functions should only be used from the compiler or operating system thread-safe libraries.

> **Tip:** Use only the main thread in a CICS application to perform **`EXEC CICS`** calls or do any extended architecture (XA) work.

Even if a function is thread-safe, you cannot always use it on the CICS environment. Functions that you can use in CICS application programs without restrictions are called CICS-safe functions.

Important: Compile any non-CICS application program that is run under the CICS environment, along with other CICS application program in the thread-safe mode. For example, if a CICS application program is linked to a thread-safe library, and is calling a non-CICS application program that links to a non-thread-safe library, such programs cause intermittent failures when run on the CICS environment.

The functions and services listed in Table 9-1 on page 414 are not CICS-safe and must *not* be used at all, or at best, used with caution. In many cases, CICS uses these functions and services itself. Their use in application programs possibly causes CICS to behave unpredictably.

Table 9-1 lists functions that you should not use with CICS.

Table 9-1 Functions and their restrictions

Function or service	Restriction
Any function that is not thread-safe	Do not use.
`exec` (without **`fork`**)	Do not use.
`setlocale`	Do not use.
Shared memory functions	Do not attach memory at the address specified with the RD RegionPoolBase attribute. CICS uses this address for region pool shared memory.
CICS internal functions	Do not use.
`exit` or **`_exit`**	Do not use.
`fork`	Do not use.
`stdin`, **`stdout`**, or **`stderr`**	Do not use these streams.
`kill`	Do not send signals to any CICS process.
`raise`	Do not use.
`assert`	Do not use.
`abort`	Do not use.
`sigprocmask`	Do not use.
`signals`	Do not use.
`signal masks`	Do not use.
`cin`, **`cour`**, or **`cerr`**	Do not use these I/O stream objects.
`TRAN`	Do not use.
Transactional C	Do not use.
`threadTid`	Do not use.
`catch(...)` (in C++ programs)	Any exceptions not generated by the application must be rethrown (using throw with no argument).

9.3.14 Considerations for programming in C and C++

The following sections provide a list of considerations that you have to be aware of when writing CICS applications using C and C++.

Restriction in cached programs using variables in static storage

When C or C++ programs are cached, variables in static storage are not reinitialized when the program is reinvoked. This is useful where database connections (among other things) are stored as variables in static storage, and are required to be kept between successive invocations because this provides a large performance benefit.

BMS maps can be stored as static, which enables previous data to be shown on the panel. If you do not want this because of the possibility of security exposures, either change the code so that the variables in static storage are not used, or do not cache the program.

Calling programs from C or C++

CICS C or C++ programs are only provided a fresh copy of their static data for the first instance of the program. You should not use static data in programs that are called recursively.

String handling

CICS does not generally support the C convention of delimiting strings using a `NULL` byte. Instead, strings are padded with spaces to their maximum length. All strings output from CICS are always space padded, and therefore cannot be used directly by C string manipulation functions.

Take care in passing strings ended with a `NULL` to the CICS interface. This might work in situations where the option concerned is not of a fixed length, but do not work for fixed-length options. For example, a CICS file name can be up to eight characters in length. Therefore, you can open the literal "`ABCD`" and CICS handles this correctly.

However, temporary storage queue names (including `REQID` on certain commands) must be exactly 8 bytes. Therefore, CICS takes whatever bytes happen to be after the null following "`ABCD`" as part of the TSQ name.

C++ considerations

Do not use I/O stream objects in CICS programs. Do not place CICS statements in header files as part of an inline function definition. Translate CICS statements in class templates before the inclusion of the template class definition in any program. C++ enables the creation of new objects on the heap, and it is important that these objects are deleted after use to avoid memory leaks. Ensure that all initialization is done within the object constructors.

Do not use CICS statements in static object constructors or destructors. Static constructors and destructors are called outside a CICS LUW, and therefore using CICS statements at these points can cause abends to occur.

9.3.15 Considerations for programming in COBOL

The following sections provide a list of considerations that you have to be aware of when writing CICS applications using COBOL.

Working storage

Applications written in COBOL are given a fresh copy of their working storage when they are first used within a transaction. It is not possible for CICS to arrange cancellation of anything but the top-level CICS program. If you want to have working storage for such programs set to the initial state, both within and for all transactions, you must ensure that these programs are canceled individually.

If you are using Micro Focus Net Express COBOL, every program is given new working storage if the `/DATA-CONTEXT` flag is specified when the program is compiled.

Recursion

Generally, applications that are written in COBOL must avoid using recursion. However, for ACUCOBOL-GT applications, CICS sets the **`RECURSION`** environment variable to `TRUE` by default. No compile flag is necessary.

Mixing languages

A run unit is a running set of one or more programs that communicate with each other by COBOL **`CALL`** statements. It is an execution of a single entry as defined in the program definitions (PD). PD entries are not required for called subprograms. In a CICS environment, a run unit is entered at the start of a CICS task, or called by a **`LINK`** or **`XCTL`** command.

Be aware of the following rules governing calls between COBOL and C or C++ programs under CICS:

- COBOL programs that contain CICS commands can **CALL** C or C++ programs if the called C or C++ programs do not contain any CICS commands.
- C or C++ programs that contain CICS commands can **CALL** COBOL programs if the called COBOL programs do not contain any CICS commands.
- COBOL programs can **LINK** or **XCTL** to a C or C++ program regardless of whether the C or C++ program contains CICS commands.

Therefore, if your COBOL program calls a C or C++ program that contains CICS commands (or vice versa), use **LINK** or **XCTL** rather than the COBOL **CALL** statement.

Passing integer data between C or C++ and COBOL

If you want to pass integer data between C or C++ and COBOL programs in a COMMAREA, the data items must be declared in COBOL as `COMP-5`, otherwise the byte ordering of the data is incorrect, and the values are corrupted.

Returning from COBOL programs

There are various methods of returning from a COBOL program:

GOBACK	Returns control to calling program or to CICS.
EXIT PROGRAM	Returns control to the calling program when issued from a subprogram started by a **COBOL CALL**. When issued from a top-level procedure division, as in a program started directly by CICS, or by **LINK** or **XCTL**, it is ignored.
STOP RUN	Do *not* use in CICS programs because it causes the application server to stop.
EXEC CICS RETURN	Ends the program normally.
EXEC CICS XCTL	Ends the program normally.
EXEC CICS ABEND	Ends the program abnormally.

Restrictions

You must not use COBOL statements to request operating system functions that can be requested from the CICS API. You should also not use the following COBOL statements:

- **STOP RUN**
- **DISPLAY**
- If the **PROGRAM-ID** statement is used, the program name must be the same as the file name. The **PROGRAM-ID** is used by the COBOL compilers as the entry-point name, defaulting to uppercase.
- On CICS on open systems, you can use compiler options to allow a mixed-case entry-point name, but CICS does not support mixed-case for Micro Focus Server Express COBOL. It supports mixed case for IBM COBOL only if you compile and link in separate steps and do not use the **cicstcl** command directly.
- On CICS for Windows, CICS does not support mixed-case entry-point names for IBM COBOL or Micro Focus Net Express COBOL.
- Micro Focus Net Express programs are never unloaded from memory. Run **CEMT SET PROG(<PROGRAM>) NEWCOPY** to unload and reload the modified program. Reloading an application will happen in each application server only during the next execution of the transaction in that application server.

9.3.16 Disabling execution diagnostic facility (EDF)

Avoid the **-e** option with the **cicstran** command to reduce processor resources used by the EDF check on every CICS API call.

9.4 System administration considerations

System administration is daily work. A good configuration can greatly reduce your workload.

9.4.1 Choice of Virtual Storage Access Method

CICS can use either the Structured File Server (SFS) system or relational database management system (RDBMS), either DB2 or Oracle, for storing VSAM files. There are advantages and disadvantages for each option. We describe these in the following sections.

Structured File Server

The following list includes advantages of using SFS:

- Easy to configure
- Requires no additional licenses
- Supports all file options, such as key-sequenced data set (KSDS), entry-sequenced data set (ESDS), and relative record data set (RRDS).
- Supports TS and TDQs
- Is recoverable

The following list includes disadvantages of using SFS:

- Requires additional skills to manage, operate, configure, and tune
- Harder to back up and restore than RDBMS

The following items are considerations for implementing SFS:

- Never perform a cold start of the SFS.

  Performing a cold start of the SFS discards all files on the SFS logical data volumes. Both the data in the files and the files themselves are lost. Each time the SFS undergoes a cold start, these files must be re-created. Therefore, cold starts are not recommended for the SFS.

- Manage SFS operational threads (`OpThreadPoolSize`).

  The SFS attribute `opThreadPoolSize` determines the maximum number of simultaneous file-control operations that can be in progress in an SFS at a time. Additional requests are queued at entry to the SFS. This is set in the Structured File Server definitions (SSD) for the SFS. The default is `12`. Set this value to be 1 more than the sum of all `MaxServer` values defined in the RD stanza for all of the regions using a particular SFS.

- Buffer pool size.

  Buffer pool size (in KB): The SFS manages an amount of storage directly, and when this fills, it hands it off page by page to the OS. If the SFS buffering is too small, it spends all of its time fetching and passing data to the OS. If the SFS buffering is too large, the OS performs excessive paging underneath the SFS. This is set in the SSD for the SFS. The default value is `1000`.

- Browse cache.

  The **`CICS_BROWSE_CACHE`** environment variable is used to specify a cache size that is to be used for browsing files on an SFS. The cache is set up in multiples of 4 KB records. Consider carefully whether to use a browse cache and, if you use one, how large a cache to use. Remember that all file browse operations use the cache. In some cases, its use improves performance, but performance can be degraded if multiple short browses are used.

Relational database management system

The advantages of using RDBMS include the following outcomes:

- Skills easily available
- Widely used and understood
- Easy to back up and restore

The disadvantages of using RDBMS include the following outcomes:

- Requires additional licenses
- Has no concept of unrecoverable data

Consider the following reasons for using IBM DB2 as the VSAM data server:

- DB2 storage space. When configuring DB2 for CICS files, the general advice is to use database-managed space (DMS) rather than system-managed storage (SMS) to manage the storage space. With DMS, DB2 manages the storage up to a predefined limit. With SMS, there are no preset limits, and DB2 uses the facilities of the file system to help manage the storage.

  DMS is generally 10% - 15% faster than SMS, but does have an increased management resource use. Setting a suitable size buffer pool in DB2 removes unnecessary disk access and enable DB2 to cache all data, in which case the difference in performance between DMS and SMS becomes minimal.

- DB2 security. Use explicit user ID and password on the database connection. If these are not defined, the authentication uses implicit credentials of service.

  Set `TP_MON_NAME` to `CICS` in DB2 to identify CICS as a transaction manager.

- Deadlocks. Using DB2 for VSAM data increases the risk of deadlocks.:
  - **`DeadLockTimeout`**. This attribute specifies, in seconds, how long CICS enables the transaction to wait when CICS has detected a possible deadlock. When that time expires, this transaction becomes a candidate for abnormal termination and receives an abnormal termination code `AKCS` if CICS chooses it. A value of `0` means that the transaction never times out. The default value is `0`.
  - **`DB2_RR_TO_RS`**. Try using the DB2 variable `DB2_RR_TO_RS` to switch off next key locking. This can avoid unwanted deadlocks, but if this variable is used, DB2 acquires more internal locks. In this case, the **`locklist`** or **`maxlocks`** database parameters possibly need to be increased to accommodate the extra internal locks. If the **`locklist`** parameter is not set sufficiently high, lock escalation occurs and concurrency suffers.

    Set the **`DB2_RR_TO_RS`** environment variable to `YES` before using **`db2start`** to start the database manager. This changes the behavior of the DB2 index manager to avoid next key locks. This can provide some performance benefits to a CICS region that uses DB2 for queue and file management.

However, setting **DB2_RR_TO_RS** to YES requires that you avoid the use of repeatable-read cursors in any DB2 program that uses this DB2 instance. A region that uses DB2 for queue and file management does not use this type of cursor. Because of this restriction, it is advisable to set this environment variable only for instances that are exclusively for CICS use.

– **locktimeout**. This parameter specifies the number of seconds that an application waits to obtain a lock. This helps to avoid global deadlocks for applications. If you set this parameter to 0, locks are not waited for. In this situation, if no lock is available at the time of the request, the application immediately receives a -911.

  If you set this parameter to -1, lock timeout detection is turned off. In this situation, a lock is waited for (if one is not available at the time of the request) until one of the following results occurs:

  - The lock is granted.
  - A deadlock occurs.

  Use benchmarking techniques to tune this parameter. Set the value to quickly detect waits that are occurring because of an abnormal situation, such as a transaction that is stalled (possibly as a result of the users leaving their workstation). You must set it sufficiently high so that valid lock requests do not time out because of peak workloads, when there is more waiting for locks.

– **dlchktime**. This is the time interval for checking the deadlock configuration parameter. The deadlock check interval defines the frequency at which the database manager checks for deadlocks among all of the applications connected to a database.

– **locklist**. This is the maximum storage for the lock list configuration parameter. This parameter indicates the amount of storage that is allocated to the lock list. There is one lock list per database, and it contains the locks held by all applications concurrently connected to the database.

  Locking is the mechanism that the database manager uses to control concurrent access to data in the database by multiple applications. Both rows and tables can be locked. The database manager might also acquire locks for internal use.

– **maxlocks**: This is the maximum percent of lock list before the escalation configuration parameter. Lock escalation is the process of replacing row locks with table locks, reducing the number of locks in the list. This parameter defines a percentage of the lock list held by an application that must be filled before the database manager performs escalation.

When the number of locks held by any one application reaches this percentage of the total lock list size, lock escalation occurs for the locks that are held by that application. Lock escalation also occurs if the lock list runs out of space.

The database manager determines which locks to escalate by looking through the lock list for the application, and finding the table with the most row locks. If, after replacing these with a single table lock, the **maxlocks** value is no longer exceeded, lock escalation stops. If not, it continues until the percentage of the lock list held is lower than the value of **maxlocks**.

- XA connections

  If only one non-SFS resource is ever updated in a transaction, consider the use of single-phase commit XA processing rather than two-phase. Use single-phase XA optimization when using a single resource manager registered with the CICS system.

  You must be aware that when using single-phase XA connections, CICS is not responsible for taking recovery actions against the XA-registered resource. The recovery has to be handled by the application itself.

  If you use non-XA connections to a database, your applications must issue the **EXEC SQL CONNECT** command to connect to the database server for each transaction. Such frequent connections use many system resources. Applications can reuse an XA-managed database connection repeatedly while the application server is running, without having to call **EXEC SQL CONNECT** repeatedly.

  DB2 supports XA dynamic registration. In this case, CICS can arrange to drive sync point in the database only when the database is actually updated for the specific transaction. This can reduce the time it takes to sync point when you are running in an environment with multiple XA databases and CICS file control files. In such an environment, transactions usually update only data that is managed by some of the applications.

Location of servers that store VSAM data

There are several options for physically locating CICS regions, SFS, Peer-to-Peer Communications (PPC) Gateways, and RDBMS servers. The simplest is to place all systems on one machine. The most complicated is to distribute all servers.

In general, CICS and SFS should always coexist on a machine, if there is no need to share SFS with multiple CICS servers. Depending on the capacity of the servers, the location of existing RDBMS servers, and the customer skill set, the RDBMS can either run on the same server as CICS or on a remote one.

9.4.2 CICS application considerations

The following sections provide a list of considerations that you have to be aware of when designing a CICS application.

User Transaction Class

You can control the number of transactions running in a region at any one time by using transaction classes (T-Classes). This is done by assigning certain types of transactions to a given transaction class, and limiting the number of transactions that can be run in that class. You can assign a transaction to a class by specifying the class attribute in the TD.

The number of transactions that can be run simultaneously in a given class is limited by using the table in the class limits attribute in the RD. CICS permits 11 transaction classes: Classes 1 - 10, and none. Only classes 1 - 10 can be controlled by using the class limits table. Transactions defined with class `none` are not subject to any class restriction. All transactions supplied by CICS have the default class of none.

Provide all user transactions a T-Class, and let the transactions supplied by CICS use none. Set `MaxServers` to be at least one greater than the sum of the values that is specified in `ClassMaxTasks`.

Security

In this section, we illustrate various techniques that you can use to protect the CICS resources.

Forcing terminal users to sign on

Always force users to authenticate using the following authentication:

- CICS Execute Sign-on (**`CESN`**)
- Custom sign-on transaction

You can use the following options to force sign-on:

- The **`-t`** option in the following commands forces an initial transaction:
- **`cicsteld`**.
- **`cicscp create telnet server`**.
- **`cicsterm / cicslterm`**.
- CICS Transaction Gateway (CICS TG) and CICS Universal Client have a similar option in the `CTG.INI` configuration file (Attribute: `INITIALTRANSID`).

Using default user

When CICS is initially configured, all of the values that define the protection levels are set to *none*. If the `AuthenticationService` attribute is set to `CICS`, do *not* change these protection levels from this default value.

> **Tip:** When the value `none` is used, users who do not have user definitions (UD) defined for them are logged on with the region's default user ID. Therefore, ensure that the default user is not given access to resources that must be protected.

Protecting CICS resources

The CICS-supplied transactions are initially defined with a standard transaction security value of `1` or `public` in the TD. This simplifies the initial installation of CICS, but has the disadvantage that it enables all CICS users to start these transactions.

Apart from running CICS Remote Transaction Execution (**`CRTE`**) to route a transaction to another system (signing on there), the users require only **`CESN`** to change their authority. For example, by default, all users can run at the same level of authority, but a user might occasionally have to use a different ID to run a secure transaction.

Transactions, such as CICS Extended Master Terminal (**`CEMT`**), CICS Execute Command-Level Interpreter (**`CECI`**), and CICS Execution Diagnostic Facility (**`CEDF`**) perform some powerful functions. Consider applying security to these transactions, and requiring specific sign-on to privileged users to run them.

For example, programmers can use the CICS Execute Check Syntax (**`CECS`**) transactions to check command syntax interactively. They can also use the **`CECI`** transaction to run commands without having to write a program. Because it is easy to change data in your system with this facility, you probably want to control the use of **`CECI`**.

In a production system, you can provide **`CEDF`** and **`CECI`**, but enable their use only with appropriate control and supervision. Use both transaction-level and resource-level security checking to control them tightly, and reserve them for the system programmer to use when extreme difficulties are encountered, and when debugging cannot be done any other way.

9.4.3 CICS region administration considerations

The following sections provide a list of administration considerations that you have to be aware of for CICS applications.

Using operating system file systems

For each CICS region or server machine, we suggest that you isolate certain directories to prevent disk space limitations from impeding the performance of your CICS system. To isolate a directory, create a separate journaled file system (JFS), or its equivalent on Solaris, Linux, and Hewlett-Packard, for it.

For a region, create a separate JFS for the `/var/cics_regions` directory. For an SFS or PPC Gateway server, create a separate JFS for the `/var/cics_servers`, `/var/cics_servers/backups`, and `/var/cics_servers/archives` directories.

Disks that hold logs are written to during every sync point. It is also advisable to put busy logs on separate drives. Otherwise, there can be a large amount of seeking to get between the database and the log area.

The CICS log directory, `/var/cics_regions/regionName/log`, holds the dynamic log data for the CICS region. The product installation automatically installs the log directory into the system's `/var` volume. If the log entries fill the `/var` volume, system performance can be affected.

A separate file system for the CICS log can be used, but it cannot be set up until after the region is configured. Because the information contained in the CICS log is required to "warm start" the region, we advise that you mirror the log so that you can recover the system to the point of failure if data is lost.

Whenever a transaction is started in a CICS region, CICS stores information about it in the CICS log. The best way of ensuring that the log data is recoverable is to use data mirroring for the file system containing the log data.

In practice, this means that the logical volume that hosts the file system must be configured for mirroring. It is best to have a dedicated file system for the log data. This gives you greater control over the allocation of file system storage, and ensures that this storage can only be used for storing CICS log data.

Important: The CICS log files are not cleared unless a region is shut down normally. Repeated abnormal shutdowns can lead to a growth in the size of the log file.

Volumes that are used as logs are written to during every sync point. It is advisable to put each of these on its own physical volume.

Otherwise, a large amount of seeking can occur to get between the database and the log area. Adding physical memory decreases the need for disk usage for purposes other than sync point.

Mirrored volumes usually schedule as many concurrent independent reads as there are mirror copies (two or three). This can increase the time for a write because the subsystem must ensure that all copies are updated.

Using MaxServers and MinServers

The following section describes how to use MaxServers and MinServers:

- Maximum application servers (`MaxServers`). This RD attribute determines the maximum number of tasks that can run simultaneously in the region.
- Minimum application servers (`MinServers`). This RD attribute determines the minimum number of application server processes that can be available in the region.

In certain cases, due to improper configuration of these two tunable attributes, there can be a frequent termination and creation of application server processes, which can result in a performance loss of the region. One of the ways to avoid the termination of the application server processes is to have the `MinServer` and `MaxServer` attributes set to the same value. This way CICS always ensures the availability of the `MinServer` and `MaxServer` value of regional application servers.

Using hash buckets

LOADed data address hash buckets is a subset of task-shared pool address hash buckets. Therefore, keep their values the same, or set task-shared pool address hash buckets one order of magnitude greater. You can then use the storage statistics for tuning both sets of tables. Table sizes greater than 512 KB can be appropriate if the region uses very large numbers of maps and tables. Both values are defined in the RD:

- `LoadDataNumBuckets` (number of LOADed data address hash buckets)

  This attribute controls access to maps and tables by specifying the size of the hash table used by CICS to find maps and tables in the task-shared pool. Set the value according to the expected usage of your region. CICS rounds up the specified value to the nearest power of two, with a minimum value of one. When setting this attribute, consider the trade-off between increased performance and increased use of storage space.

  You can tune the value of this attribute by using statistical information. For further information about statistics, see the task about Administering in the TXSeries documentation library (the default value is 512):

  http://www.ibm.com/support/knowledgecenter/SSAL2T/welcome

- `TaskSHNumBuckets` (number of task-shared pool address hash buckets)

  This attribute controls access to the task-shared pool by specifying the size of the hash table used by CICS to find items in this pool. Set the value according to the expected usage of your region. CICS rounds up the value that you specify to the nearest power of two, with a minimum value of `512`. The value you supply depends on the expected usage of your region.

  When setting this attribute, consider the trade-off between higher performance and increased use of storage space. You can tune the value of this attribute by using statistical information. For further information about statistics, see the task about Administering in the TXSeries documentation library. The default value is `512`.

Using main over auxiliary for temporary storage queues

The main temporary storage is unrecoverable, and is allocated from region storage. Records in the auxiliary temporary storage are kept in a file system. The queue pointers are kept in the main storage, and can be saved in the file system (for recoverable temporary storage queues) at sync point time. If application logic and available memory allow, using the main storage rather than auxiliary temporary storage helps to maximize performance.

Temporary data can be stored either in the main storage (memory) or in the auxiliary storage (a database file). Generally, use the main storage if the data is required for short periods of time. Use the auxiliary storage if the data is to be kept for long periods of time.

Data stored in auxiliary recoverable temporary storage queues is retained after CICS termination, and can be recovered in a subsequent restart. Data stored in auxiliary unrecoverable temporary storage queues is retained only for a normal shutdown, but not for an immediate shutdown or system failure unless a database is being used as the File Manager. Data stored in the main storage is not retained across any type of shutdown and so cannot be recovered.

Using unrecoverable resources

When recoverable resources are accessed for updating, extra logging is required during synchronization, which incurs more I/O and processor usage. Because of this, define resources as recoverable only if they really have to be. In particular, if you use files, TDQs, and TSQs that are read-only, define them as unrecoverable. You can back them up separately in case of a system or disk failure.

Cache programs

Programs in C, C++, IBM COBOL, or PL/I are cached only if at the time a program is loaded, the `PD Resident` attribute is set to `yes`, and the number of cached programs in use has not reached the maximum number. Because individual programs in use are not removed from the cache, it is advisable that the cache size enables programs at every logical level to be cached when **`LINK`** is used. Increase the **`ProgramCacheSize`** value in the RD from the default of `0` (zero).

Use the program caching feature if your programs are repeatedly called. If you are coding in C, and you use static data in programs and do not initialize before use, you should not use the caching feature.

The **`ProgramCacheSize`** (maximum number of C or IBM COBOL programs that can be cached) attribute specifies the maximum number of programs that can be in one application server's program cache. A program is cached at the time it is loaded only if the `PD Resident` attribute is set to `yes` and the number of cached programs in use has not reached the maximum number. A resident program is not loaded again until one of the following things occurs:

- A **`SET PROGRAM COPY(NEWCOPY)`** or **`SET PROGRAM NEWCOPY`** command is issued.
- The value of the `PD Resident` attribute is changed to `no`.
- The cache is full and the program is removed to make room for a new entry. In this case, the least recently used program is removed, leaving more frequently used programs in the cache.

Because programs that are being used are not removed from the cache, it is advisable that the cache size enables programs at every logical level to be cached when using **`LINK`**. The default value is `0` (zero).

Using full path name

The `PathName` (program path name) attribute specifies the directory path and file name for the program, map set, or table. The default value is "" (null). You can specify the path in three ways:

- An absolute path name to the file, followed by the file name
- A relative path name to the file, followed by the file name:

  CICS interprets the path name as relative to `/var/cics_regions/<regionName>/bin` (for distributed systems) or `rootDir\var\cics_regions\<regionName>\bin` (for Windows).
- A file name string not containing any slash character (/) (for open systems) or backslash character (\) (for Windows)

To find a program specified as a string, CICS searches the directories in the following order:

1. `/var/cics_regions/<regionName>` (for open systems) or `rootDir\var\cics_regions\<regionName>` (for Windows)
2. `$CICSPATH/bin` (for open systems)

For CICS to find a map set specified as a string, the file name must have the extension `.map`. CICS searches the directories in the following sequence:

1. `/var/cics_regions/<regionName>/maps/<locale>` (for open systems) or `rootDir\var\cics_regions\<regionName>\maps\<locale>` (for Windows)
2. `/var/cics_regions/<regionName>/maps/prime` (for open systems) or `rootDir\var\cics_regions\<regionName>\maps\prime` (for Windows)
3. `$CICSPATH/maps/<locale>` (for open systems) or `rootDir\opt\cics\maps\<locale>` (for Windows)
4. `$CICSPATH/maps/prime` (for open systems) or `rootDir\opt\cics\maps\prime` (for Windows)

In these cases, `<locale>` is the language specified for the system.

You can specify the user-defined path through the environment variable **CICS_PROGRAM_PATH**. The **CICS_PROGRAM_PATH** environment variable supports standard operating system behavior for paths. Therefore, on Windows platforms, the paths are delimited with a semi-colon (;), and on UNIX platforms, they are delimited with a colon (:).

When a `PD PathName` attribute specifies a file name that does not contain any forward slash (/) for open systems, or backslash (\) for Windows, and the program cannot be found in the locations that are mentioned previously, CICS searches the paths mentioned to try to load the program or map. For this search, CICS uses the **CICS_PROGRAM_PATH** environment variable and the `PathName` attribute.

If the **cicsteld** command is used to connect users, the **-l** option can also be used to specify the language. For information about specifying the language, see the *Installing* and *Administering* tasks in the TXSeries documentation library, for your operating system.

To find a table specified as a string, CICS searches the directory `/var/cics_regions/<regionName>/data` (for open systems) or `rootDir\var\cics_regions\<regionName>\data` (for Windows). For programs written in a language other than Java, the file name specified in the `PathName` attribute might not contain an extension. For programs written in Java, the file name must include the `.class` suffix.

On CICS for distributed systems when Micro Focus Server Express COBOL Animator is not used (not all languages are available on all operating systems), CICS searches for the program file based on its installed extension in the following orders:

1. C program files (no extension)
2. IBM COBOL files (with extension `.ibmcob`)
3. Micro Focus Server Express COBOL native code files (with extension `.gnt`)
4. Micro Focus Server Express COBOL intermediate code files (with extension `.int`)
5. IBM PL/I files (with extension `.ibmpli`)
6. IBM C++ files (with extension `.ibmcpp`)

On Windows systems, CICS searches for the program file based on its installed extension in the following orders:

1. Micro Focus Net Express COBOL files (with extension `.cbmfnt`, `.int`, or `.gnt`). See the note following this list.
2. C program files (with extension `.dll`).
3. IBM COBOL files (with extension `.ibmcob`).
4. IBM PL/I files (with extension `.ibmpli`).
5. IBM C++ files (with extension `.ibmcpp`).
6. Microsoft C++ files (with extension `.cpp`).

The order by which CICS searches through Micro Focus files depends upon the following conditions:

- Whether you are using Micro Focus COBOL or Net Express
- If you are using Micro Focus COBOL, whether you are using Animator

If you are using Micro Focus Net Express COBOL on Windows, CICS searches for Micro Focus Net Express files in this order whether you are using Animator:

1. Micro Focus Net Express COBOL generated code files (with extension `.cbmfnt`)
2. Micro Focus Net Express native code files (with extension `.gnt`)
3. Micro Focus Net Express COBOL intermediate code files (with extension `.int`)

If you are using Micro Focus Server Express COBOL with Animator on UNIX, CICS searches for Micro Focus files in this order:

1. Micro Focus Server Express COBOL intermediate code files (with extension `.int`)
2. Micro Focus Server Express COBOL native code files (with extension `.gnt`)

If you are using Micro Focus Server Express COBOL without Animator on UNIX, CICS searches only for Micro Focus files with the extensions `.gnt` and `.int`. If you are using Micro Focus Net Express COBOL without Animator on Windows, CICS searches only for Micro Focus Net Express files with the extension `.cbmfnt`.

Using SafetyLevel

The CICS internal structures in the region pool are protected from rogue applications by the safety level defined in the `SafetyLevel` attribute in the RD stanza. Possible values are `normal`, `none`, and `guard`. If you set this attribute to `normal`, the CICS region pool cannot be accessed by user programs. This incurs a significant performance cost.

If you set this attribute to `guard`, only the first and last pages of the CICS region pool are protected, with less of a performance cost. If you set this attribute to `none`, the CICS region pool is not protected, and you gain in performance at the cost of a potential loss of integrity. The default value is `none`. Run with `normal` only if you experience problems with storage corruption.

Note: The `SafetyLevel` setting is ignored on CICS for Solaris systems, and user transactions can always access the region pool.

The `guard` setting is provided specifically for use on the Windows operating system, on which the normal `SafetyLevel` setting incurs significant performance costs. To lessen these performance costs, set `SafetyLevel` to `guard`, which causes only the first and last pages of the region pool to be protected. On CICS for AIX and CICS for Hewlett-Packard UNIX (HP-UX) systems, the `guard` setting is treated the same as the `normal` setting. The default value is `none`.

Using timeouts

Use transaction **`Timeout`** and **`DeadLockTimeout`** parameters in TD stanza to ensure transaction completion. This attribute is only effective if not in the application program:

- **`DeadLockTimeout`** (transaction deadlock timeout value (seconds)). This attribute specifies, in seconds, how long CICS enables the transaction to wait when CICS has detected a possible deadlock. When this time expires, this transaction becomes a candidate for abnormal termination, and receives an abnormal termination code **`AKCS`** if CICS chooses it. A value of `0` means that the transaction never times out. The default value is `0`.
- **`Timeout`** (conversational timeout value (minutes)). This attribute specifies, in minutes, how long CICS waits for terminal input when running this transaction. When this time expires, this transaction abnormally ends with the code AKCT. The value 0 means that the transaction never times out. The default value is 0.

> **Important:** The value set for the timeout attribute overrides the setting for the `XPRecvTimeout` region definition attribute.

Increasing IntrospectInterval

After your CICS system is stable, increase the `IntrospectInterval` value to, for example, one time in 2 hours. CICS schedules a private transaction known as *CLAM*, which checks the internal consistency of the CICS region. To enable the consistency check, set the following attributes in the RD stanza:

- **`IntrospectInterval`** (interval between region consistency checks (minutes)). This attribute specifies, in minutes, the time interval between periodic checks to monitor the integrity of a region. You can specify the value in the range of `0` (zero) - 1440. The value `0` switches off integrity checking. Checking integrity at frequent intervals decreases performance. The default value is `10`.
- **IntrospectLevel** (level of checking to perform on region). Currently, this attribute has a fixed value of `minimal`.

> **Tip:** In CICS regions that are used for development purposes, always set the `IntrospectInterval` to a lower value, forcing CICS to do the internal consistency check of the CICS region frequently. This is particularly useful when the application program in its operation might corrupt certain parts of the shared memory being used by the application and CICS together.

Minimizing intersystem communication

Intersystem communication (ISC) includes the following facilities:

- Function shipping
- Transaction routing
- DPL
- Distributed transaction processing (DTP)
- Asynchronous starts

All of the ISC facilities use resources and performance. Try to minimize using the ISC facilities and, where possible, use local resources.

Managing startup and shutdown program lists

The following sections describe how to manage the startup and shutdown program list.

Starting transactions automatically after region start

In some real implementation scenarios, it is required to start some transactions or programs immediately after the region start, such as user monitor transactions. This can be enabled by setting the `StartupProgList` attribute in the RD stanza with the program name to be called during the region startup.

The `startprg.ccs` source program included with this book provides a demonstration of how to write a startup program to start transactions that you want to start automatically after region start. To enable this program for your region, perform the following steps:

1. Compile the program with the following command:

   ```
   cicstcl -lC startprg.ccs
   ```

2. Copy the executable file to the region's bin directory:

   ```
   cp startprg /var/cics_regions/$CICSREGION/bin
   ```

3. Install the program:

   ```
   #cicsadd -c pd -r <region name> STARTPRG \
   PathName="startprg" RSLKey=public
   ```

   In the scenario of this book, use the region name `txdemo`.

4. Update the region definition:

   ```
   cicsupdate -c rd -r <region name> StartupProgList="STARTPRG"
   ```

   In the scenario of this book, use the region name `txdemo`.

5. Define transactions that you want to start in the region environment variable with a colon (:) as the separator. This program can support a maximum of 10 transactions in the list:

   ```
   START_TRANS_LIST=TRANS1:TRANS2
   ```

   In the scenario of this book, use the transation **CMTM**.

   > **Important:** The **CMTM** transaction is an alternative user application to **CKTI**, to monitor the WebSphere MQ initiation queue for incoming messages and react as required. It is a long-running transaction and cannot shut itself down when region shutdown occurs.
   >
   > You must purge this transaction before you shut down the region, or you have to shut down region with the **-f** option. An alternative solution to avoid this complex operation is introduced in the next section.

   Next, we compile the application for the **CMTM** transaction.

6. Compile the `cmqtrig.ccs` sample application:

   ```
   # cicstran -lC cmqtrig.ccs
   # xlc_r -emain -bI:/usr/lpp/cics/lib/cicsprC.exp \
   -I/usr/lpp/cics/include \
   -I/usr/mqm/inc \
   -L/usr/mqm/lib -lmqm_r -L/usr/lpp/cics/lib \
   -ocmqtrig cmqtrig.c
   ```

7. Copy the program to the region binary directory:

   ```
   #cp cmqtrig /var/cics_regions/txdemo/bin
   ```

8. Update the CICS region database:

   ```
   #cicsadd –c pd –r txdemo CMQTRIG PathName="cmqtrig" RSLKey=public
   #cicsadd –c td –r txdemo CMTM ProgName="CMQTRIG" RSLKey=public
   RSLCheck=none
   ```

9. Add the following environment variable in the region environment file:

   ```
   MQ_INIT_QUEUE=CICS.INITQ
   MQ_QMGR=mqdemo
   ```

10. Cold start the region. You can see the output in the console file, as shown in Example 9-1.

Example 9-1 Implementation of StartupProgList

```
ERZ010063I/0070 11/23/05 17:20:44.037527474 txdemo 1798364/0001
: Running startup program 'STARTPRG'
<START PROGRAM> Start transaction CMTM succeed.
ERZ010120I/0243 11/23/05 17:20:44.038872560 txdemo 1798364/0001
: Completed processing startup or shutdown program list
```

Note: You can use the startup program functionality to preinstall and cache all of your important business applications in all of the application servers. Therefore, when your business starts, CICS application servers do not have to spend time loading the application program, which increases the performance of the region.

Purging transactions automatically before region shutdown

As described previously, long-running transactions, such as **CMTM**, cannot stop themselves before region shutdown. CICS region does not shut down until all the user tasks running in that region is completed. It is not practical to purge all of the user tasks and then shut down the CICS region, because the number of tasks running in that region might be too high.

One way to resolve this problem is to write a shutdown program, which inquires on all the running tasks in that region, and purges them with the **SET FORCEPURGE** command. The shutdown program is automatically called by the CICS region during the shutdown processing, and the program name can be set in the ShutdownProgList1 attribute in RD stanza.

The `purge.ccs` source program included with this book provides a demonstration. To enable this program for your region, perform the following steps:

1. Compile the program with the following command:

   ```
   cicstcl -lC purge.ccs
   ```

2. Copy the executable file to the region's bin directory:

   ```
   cp purge /var/cics_regions/$CICSREGION/bin
   ```

3. Install the program:

   ```
   #cicsadd -c pd -r <region name> PURGE \
   PathName="purge" RSLKey=public
   ```

   In the scenario of this book, use the region name `txdemo`.

4. Update the region definition:

   ```
   cicsupdate -c rd -r <region name> ShutdownProgList1="PURGE"
   ```

   In the scenario of this book, use the region name `txdemo`.

5. Perform a cold start of the region, and after some time, shut down the region. You can see messages appearing in the console similar to those shown in Example 9-2.

Example 9-2 Implementation of ShutdownProgList1

```
ERZ010109I/0070 11/23/05 17:21:22.483580629 txdemo 1052702/0001
: Running shutdown program 'PURGE', from shutdown program list 1
TRAN: CEMT TASKNUM: 8 PURGING

TRAN: CRAB TASKNUM: 10 PURGING

TRAN: CMTM TASKNUM: 6 PURGING

TRAN: CEMT TASKNUM: 8 PURGING

TRAN: CRAB TASKNUM: 10 PURGING

TRAN: CMTM TASKNUM: 6 PURGING

ERZ010120I/0243 11/23/05 17:21:32.504933720 txdemo 1052702/0001
: Completed processing startup or shutdown program list
ERZ010035I/0042 11/23/05 17:21:32.506809863 txdemo 1097762/0001
: Waiting for user tasks to complete
ERZ014040E/0107 11/23/05 17:21:45.453609287 txdemo 1253546/0001
RIJH: First abend 'A147' occurred for transaction 'CEMT' in program
'DFHCEMT'
ERZ014016E/0028 11/23/05 17:21:45.453870535 txdemo 1253546/0001
RIJH: Transaction 'CEMT', Abend 'A147', at 'RIJH'.
SERVICE_MESSAGE 11/23/05 17:21:45.454004279 txdemo 1253546/0001
: Abend 'A147' (first abend 'A147') is reported as transaction
'CEMT' is purged.
SERVICE_MESSAGE 11/23/05 17:21:45.454115469 txdemo 1253546/0001
: Abend 'A147' (first abend 'A147') is reported as transaction
'CEMT' is force-purged.
ERZ016050W/0234 11/23/05 17:21:45.455008702 txdemo 1253546/0001
RIJH: Logical unit of work for transaction 'CEMT' has been backed
out; Distributed Transaction Service (TRAN) reason 'ENC-tra-1025: A
client (not the transaction service) aborted'
```

10

Enable IBM TXSeries applications as JSON web services for mobile workloads

This chapter introduces approaches to expose IBM TXSeries applications (apps) to the mobile space. You can leverage enterprise services and develop new services for building JavaScript Object Notation (JSON) web services for TXSeries Customer Information Control System (CICS) programs using IBM CICS Transaction Gateway (IBM CICS TG) V9.1.

IBM TXSeries for Multiplatforms has been capable of hosting mobile application services since version 7.1. Customers can use the Inbound SOAP SupportPac that was introduced in version 7.1 to build service-oriented architecture (SOA)-based solutions, and therefore mobile solutions.

CICS TG V9.1 introduces a new feature of enabling Hypertext Transfer Protocol (HTTP) clients to start communication area (COMMAREA) or channel-based CICS programs using JSON data. As shown in Figure 10-1 on page 438, this feature enables additional capabilities for TXSeries. TXSeries CICS programs can be exposed as JSON-based and RESTful web services, further enhancing the options for enterprise applications to mobile devices.

Figure 10-1 shows TXSeries mobile access.

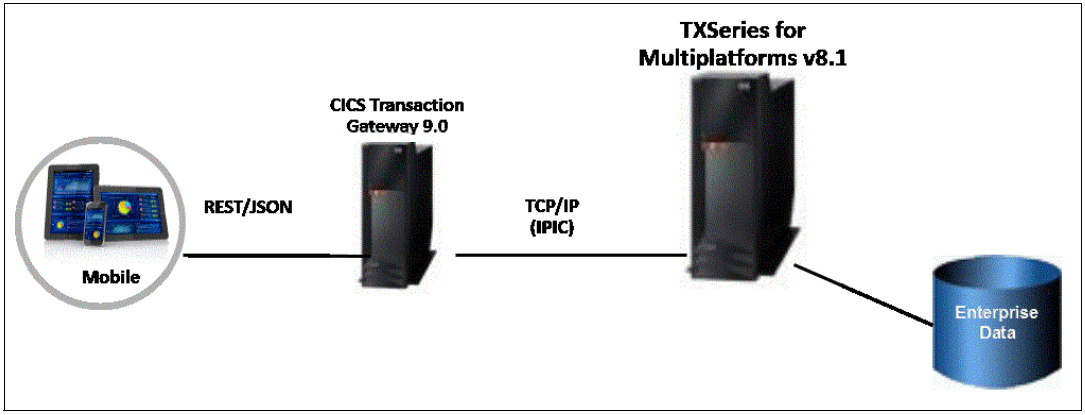


Figure 10-1 Accessing TXSeries Applications as JSON web services

This chapter provides information about the following topics:

10.1 Introduction

This section gives you a brief overview of the different technologies that expose your enterprise services on TXSeries to the mobile space.

10.1.1 TXSeries support for mobile technologies

The following paragraphs describe the main mobile technologies.

SOAP web services

SOAP is an Extensible Markup Language (XML)-based protocol for application-to-application information exchange. SOAP can be used to create `request-response` interactions. SOAP is a lightweight protocol that is independent across platforms, operating systems (OS), and transports.

TXSeries for Multiplatforms has supported SOAP-based inbound web services capability with the Inbound SOAP SupportPac on version 7.1. This feature has now been internalized and provided ready-to-use in version 8.1.

JavaScript Object Notation and Representation State Transfer

JSON is an open standard format for data interchange. JSON has a simple format with two structures, *objects* and *arrays*. Objects are an unordered collection of name-value pairs. Arrays are ordered sequences of values. JSON supports four simple types:

- String
- Number
- Boolean
- Null

JSON can be considered both human-readable and machine-readable.

Representational State Transfer (REST) is a defined set of architectural principles by which you can design web services that focus on service resources. REST is modeled around the resources in the system. Each resource is globally identifiable through its Uniform Resource Identifier (URI). Any action on a resource is defined based on the existing HTTP methods:

GET	Retrieve a resource representation.
PUT	Modify a resource representation.
POST	Create a new resource representation.
DELETE	Delete a resource representation.
HEAD	Retrieve a resource's metadata.

10.1.2 TXSeries JSON web services using CICS TG V9.1

CICS TG V9.1 introduces a new feature that enables TXSeries CICS to receive HTTP web service requests using the JSON data format. TXSeries can be configured so that your existing applications can take advantage of this without any need for the application to be updated. This configuration can be driven from either a JSON schema, where CICS generates your high-level data structure format, or from the structure to a JSON schema.

Using this feature, an HTTP payload in JSON data format is processed, and converts the data into the high-level language structure of a target TXSeries application. The data is then passed to the CICS application in either *channels and containers* or COMMAREA.

When configured from a JSON schema, you can also call your TXSeries applications in a RESTful architectural style. You can configure your JSON web services to call several different CICS programs based on the HTTP method used to make the call. The CICS programs then also receive further information, such as the query string that was used to make the RESTful request. This information can then be used in the logic of your CICS applications.

10.2 Solution architecture

The mobile solution architecture to access TXSeries web services can be built in two ways, directly and indirectly. In the direct method, the mobile devices access the web services as exposed by TXSeries for SOAP-based web services, and TXSeries and CICS TG for JSON web services.

This method, shown in Figure 10-2, is more applicable when the connecting mobile devices are trusted, and the application does not need frequent updates.

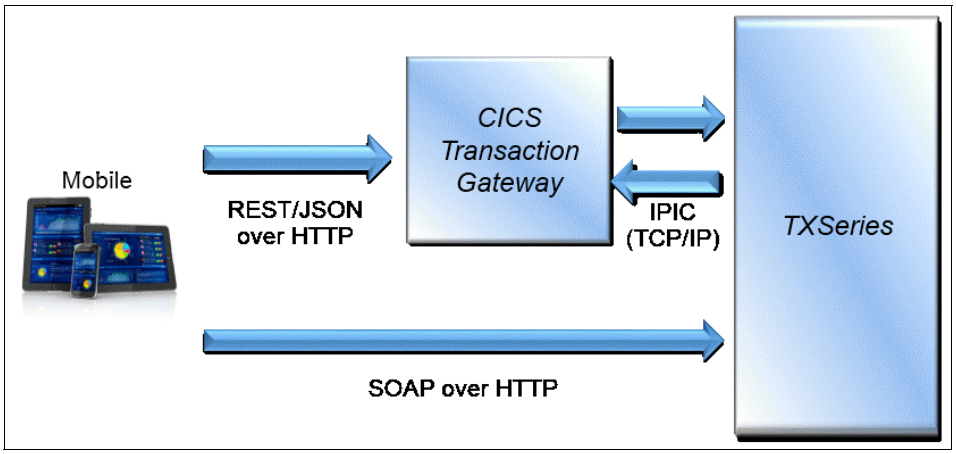


Figure 10-2 Access TXSeries web services directly from mobile

The second, indirect, approach is to use IBM Worklight®, as shown in Figure 10-3 on page 441. This approach creates a new tier, the Worklight server, between the mobile device and the exposed TXSeries CICS web service to handle devices' specific details. Worklight is a solution that provides governance and security for your mobile apps, and a powerful software development kit (SDK) for rapid development of your enterprise apps on most platforms.

Worklight also manages the creation of application versions. This functionality enables new versions of the application to be created without the need for multiple versions of the back-end business applications with logic to handle the different requests. On the different platforms, features, such as notifications, are also handled and standardized by Worklight.

Figure 10-3 shows the indirect approach with a Worklight server.

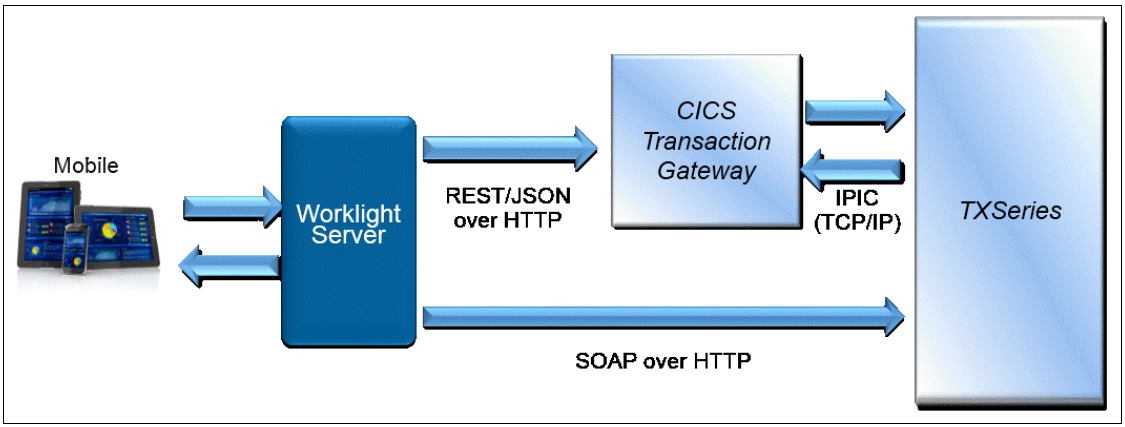


Figure 10-3 Access TXSeries web services through a Worklight server

We have seen how TXSeries can be enabled for SOAP web services using the inbound SOAP support described in 4.6.1, “Accessing TXSeries web services using the native inbound SOAP support” on page 166. This chapter focuses on enabling JSON web services for TXSeries with CICS TG V9.1.

10.2.1 Approaches to enabling JSON web services with TXSeries

You can take two approaches to building JSON-based web services for mobile:

- Top-down
- Bottom-up

The top-down approach

This method of building new enterprise services for a mobile application lends itself to the RESTful architectural style. In this style, the target resource, and the operation to be performed against it, are defined by a combination of a well-structured URI and one of the four HTTP methods (Get, Post, Put, and Delete).

The top-down approach, shown in Figure 10-4, enables you to create a set of services with a concise interface.

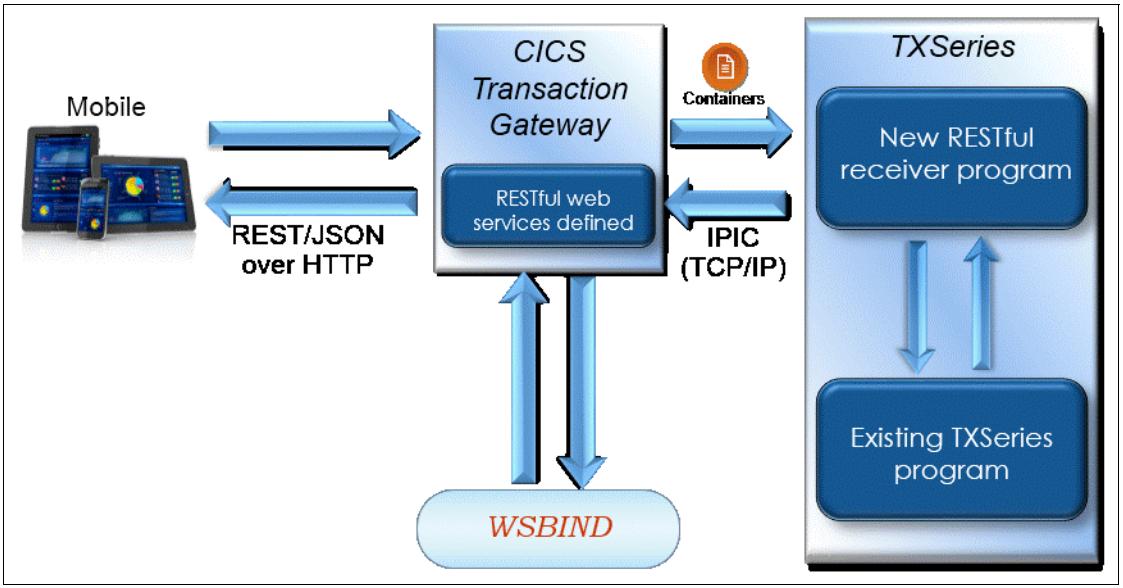


Figure 10-4 Top-down approach

The bottom-up approach

This is a faster approach for delivering enterprise services to mobile devices. Building on an existing SOA, the bottom-up approach enables you to define a JSON or SOAP interface to an existing COBOL, C, or PL/I application. This approach, shown in Figure 10-5, maximizes the reuse of existing assets, and minimizes the creation of new components.

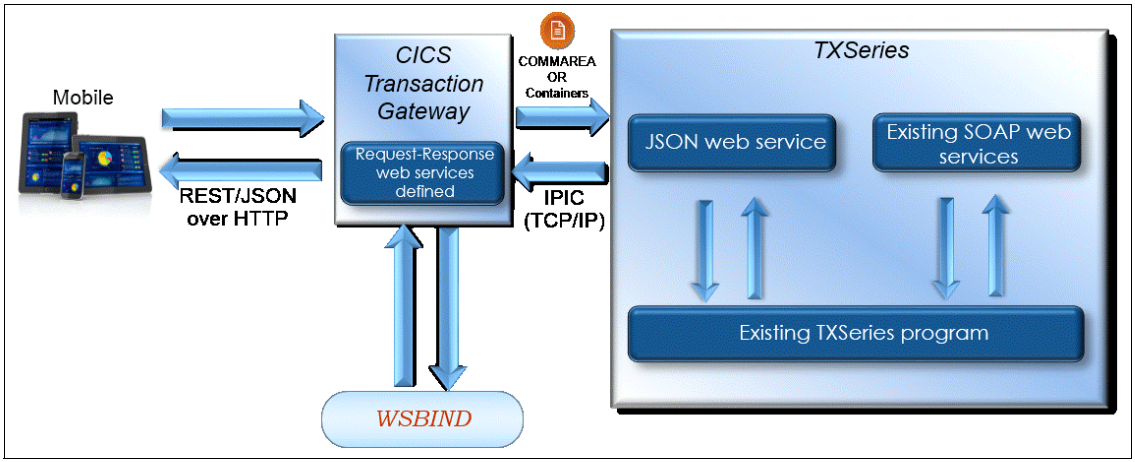


Figure 10-5 Bottom-up approach

10.2.2 A Big picture

An enterprise mobile solution using TXSeries web services can be deployed in a product stack with the following IBM products:

- IBM Worklight is a hybrid mobile solution offering governance and a powerful SDK to build applications with a server component that will drive the future mobile world.
- IBM DataPower® enables you to secure, integrate, and optimize SOA capabilities that scale.

Figure 10-6 shows a possible architecture for an enterprise mobile solution that uses TXSeries services.

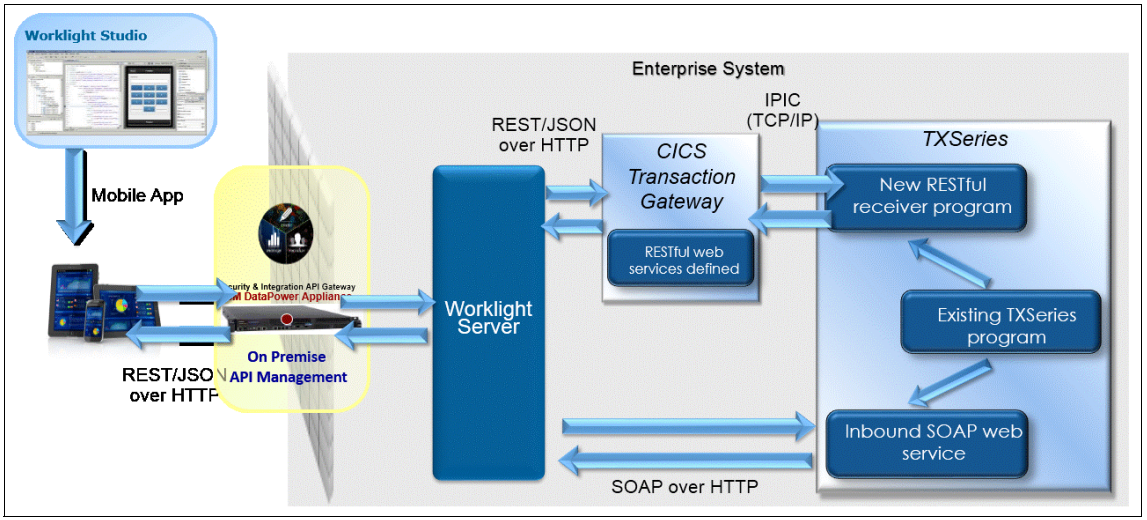


Figure 10-6 The big picture

10.3 Setting up and configuring an existing TXSeries program as a JSON web service

This section takes you through the steps to enable an existing TXSeries program as a JSON web service. We use the top-down approach, where we start with an existing TXSeries program. This program is used to generate a JSON schema. This schema can then be used by a mobile application to create JSON-based requests to call TXSeries services. Creating a mobile application is outside the scope of this book.

To enable a TXSeries program as a JSON web service, complete the following steps:

1. Configure the TXSeries region for Internet Protocol interconnectivity (IPIC).
2. Generate JSON schema and Berkeley Internet Name Domain (BIND) files using web service assistant tools from CICS Transaction Gateway.
3. Configure CICS Transaction Gateway for JSON web services.

10.3.1 Reviewing the sample TXSeries program

We use a simple CICS Common Business Oriented Language (COBOL) program, a *hello world* program for our purposes. This program is a COMMAREA-based program that reads the COMMAREA, sets it to the value “Hello, World!” and returns it back. Our goal is to be to send a JSON request with “Hello, TXSeries” as its content, and receive a JSON response with “Hello, World!” as its content.

Figure 10-7 shows the sample COBOL program. Figure 10-8 has the corresponding copybook, which is a single field called `HELLOFIELD`. The copybook has to be separated from the program because we will use the copybook later to generate corresponding JSON schema files.

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID.   HELLO.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        01 TEXT-OUT    PIC X(15) VALUE 'Hello, World!'.

        LINKAGE SECTION.
        01 DFHCOMMAREA.
           COPY HELLOCOB.

        PROCEDURE DIVISION
            USING DFHCOMMAREA.
            DISPLAY "PROGRAM TO CHECK " TEXT-OUT.
            IF EIBCALEN EQUAL 0
               DISPLAY "COMMAREA LENGTH IS ZERO"
            ELSE
              MOVE TEXT-OUT TO DFHCOMMAREA
              DISPLAY "MODIFIED COMMAREA:" DFHCOMMAREA
            END-IF.
           EXEC CICS RETURN END-EXEC
           GOBACK.
```

Figure 10-7 Sample COBOL program to expose as JSON web service

```
# cat HELLOCOB.cpy
          03 HELLOFIELD PIC X(32700).
```

Figure 10-8 COBOL copybook corresponding to the sample COBOL program

10.3.2 Configure the TXSeries region

We need to configure the region for IPIC protocol for communication between CICS Transaction Gateway and the TXSeries region. To configure the TXSeries region, complete the following steps:

1. Create the region, as shown in Figure 10-9.

```
# cicscp -v create region txdemo
ERZ058504I/0107: Starting RPC daemon.
ERZ058502I/0101: RPC daemon is already running.
ERZ096118I/0247: Creating a region.
ERZ046045I/0222: Creating region 'txdemo' from archive o
f default region '/usr/lpp/cics/DEFAULT'
ERZ010114I/0728: Creating subsystem 'cics.txdemo' for re
gion 'txdemo'
ERZ046339I/0421: Successfully imported region 'txdemo'
ERZ096111I/0224: Processing a start sfs_server command.
ERZ096141I/0224: Starting SFS server '/.:/cics/sfs/augus
t'.
ERZ096112I/0229: The SFS server '/.:/cics/sfs/august' is
 running.
ERZ096113I/0231: SFS server has '/.:/cics/sfs/august' su
ccessfully started.
ERZ038176I/0339: Adding TSQ file 'txdemocicsrectsqfile'
to server '/.:/cics/sfs/august', volume 'sfs_Saugust'.
ERZ038176I/0344: Adding TSQ file 'txdemocicsnrectsqfil'
to server '/.:/cics/sfs/august', volume 'sfs_Saugust'.
ERZ038177I/0349: Adding TDQ file 'txdemocicstdqlgfile' t
o server '/.:/cics/sfs/august', volume 'sfs_Saugust'.
ERZ038177I/0354: Adding TDQ file 'txdemocicstdqphfile' t
o server '/.:/cics/sfs/august', volume 'sfs_Saugust'.
ERZ038177I/0359: Adding TDQ file 'txdemocicstdqnofile' t
o server '/.:/cics/sfs/august', volume 'sfs_Saugust'.
ERZ038178I/0364: Adding Local Queueing file 'txdemocicsn
lqfile' to server '/.:/cics/sfs/august', volume 'sfs_Sau
gust'.
ERZ038178I/0369: Adding Local Queueing file 'txdemocicsp
lqfile' to server '/.:/cics/sfs/august', volume 'sfs_Sau
gust'.
ERZ010013I/0024: CICS has removed the lock file for regi
on 'txdemo'
ERZ096121I/0256: The region 'txdemo' was created success
fully.
ERZ096002I/0003: The cicscp command has completed succes
sfully.
#
```

Figure 10-9 Creating a TXSeries CICS region txdemo

2. Define a new program in the TXSeries region program definition (PD) stanza using the **cicsadd** command, as shown in Figure 10-10.

```
#cicsadd -c pd -r txdemo HELLO PathName="/var/cics_regio
ns/txdemo/bin/HELLO.ibmcob" RSLKey="public"
#
```

Figure 10-10 Define a new program

3. Define a new listener definition in the TXSeries region for IPIC protocol, as shown in Figure 10-11.

> **Information:** The "TXDEMO" service is a service defined in the /etc/services file, and is mapped to 20003/tcp.

```
#cicsadd -c ld -r txdemo JSONHELO TCPAddress="angela.in.
ibm.com" TCPService="TXDEMO" Protocol=IPIC
#
```

Figure 10-11 Define an IPIC listener

4. Copy the compiled program, HELLO.ibmcob, to the region bin path, as shown in Figure 10-12. Start the TXSeries region.

```
# cicscp -v start region txdemo
ERZ058504I/0107: Starting RPC daemon.
ERZ058502I/0101: RPC daemon is already running.
ERZ096122I/0264: Processing the 'start region' command.
ERZ096158I/0264: The region 'txdemo' is starting.
ERZ096111I/0224: Processing a start sfs_server command.
ERZ096141I/0224: Starting SFS server '/.:/cics/sfs/augus
t'.
ERZ096112I/0229: The SFS server '/.:/cics/sfs/august' is
 running.
ERZ096113I/0231: SFS server has '/.:/cics/sfs/august' su
ccessfully started.
ERZ038216I/0382: Subsystem 'txdemo' has been initialized
.
ERZ038219I/0390: Server 'txdemo' is responding to RPCs.
ERZ096126I/0276: Region 'txdemo' has successfully starte
d.
ERZ096002I/0003: The cicscp command has completed succes
sfully.
#
```

Figure 10-12 Start the region

10.3.3 Generate JSON Schema and BIND files

The next action is to generate the necessary artifacts for conversion between a COBOL copybook and a JSON structure. CICS Transaction Gateway provides a web services tool called **ctgassist** to generate these artifacts:

1. The web services assistant tool must be provided with an input parameter file, which provides the properties of the newly created web services. Figure 10-13 on page 448 shows the input parameter file used for this sample. A detailed explanation of each field is available here:

   http://www.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSZHFX_9.1.0/progguide/topics/json_assistant.html

```
LANG=COBOL
PGMNAME=HELLO
PGMINT=COMMAREA
URI=/hello
LOGFILE=/redbook/HELO.log
MAPPING-MODE=LS2JS
WSBIND=/redbook/HELO.wsbind
LS-REQUEST=/var/cics_regions/txdemo/bin/HELLOCOB.cpy
LS-RESPONSE=/var/cics_regions/txdemo/bin/HELLOCOB.cpy
STRUCTURE=(HELLOCOB,HELLOCOB)
DATA-TRUNCATION=ENABLED
TARGET-CICS-PLATFORM=LinuxI
JSON-SCHEMA-REQUEST=/redbook/HELO-REQ.json
JSON-SCHEMA-RESPONSE=/redbook/HELO-RES.json
```

Figure 10-13 Input parameters in a file called LS2JS_HELLO

2. Supply the input parameter file to the **`ctgassist`** command, as shown in Figure 10-14.

```
# ctgassist LS2JS_HELLO
CTGAssist - CICS Transaction Gateway JSON web services a
ssistant
(C) Copyright IBM Corporation 2014.  All rights reserved
.
CTG8733I The JSON web services assistant completed succe
ssfully
#
```

Figure 10-14 The ctgassist command

This results in the generation of a web services BIND (WSBIND) file and two JSON schema files, corresponding to a request and response. The WSBIND file is used internally for mapping between JSON requests to the COMMAREA content that the TXSeries region expects.

3. The JSON Schema file can be used by the mobile application to create a new JSON payload. Figure 10-15 shows the generated schema for a request, `HELO-REQ.json`.

```
# cat HELO-REQ.json
{
   "$schema":"http:\/\/json-schema.org\/draft-04\/schema
#",
   "description":"Request schema for the HELLO JSON inte
rface",
   "type":"object",
   "properties":{
      "HELLOOperation":{
         "type":"object",
         "properties":{
            "hellofield":{
               "type":"string",
               "maxLength":32700
            }
         },
         "required":[
            "hellofield"
         ]
      }
   },
   "required":[
      "HELLOOperation"
   ]
}#
```

Figure 10-15 HELO-REQ.json

10.3.4 Configure CICS Transaction Gateway for JSON web services

We are now ready to configure CICS Transaction Gateway. These are the final steps in the process:

1. Start the **ctgcfg** graphical user interface (GUI) tool to create a new server, as shown in Figure 10-16. This server has to connect to the TXSeries region over IPIC protocol. Ensure that the Port number matches with the port on which TXSeries listens.

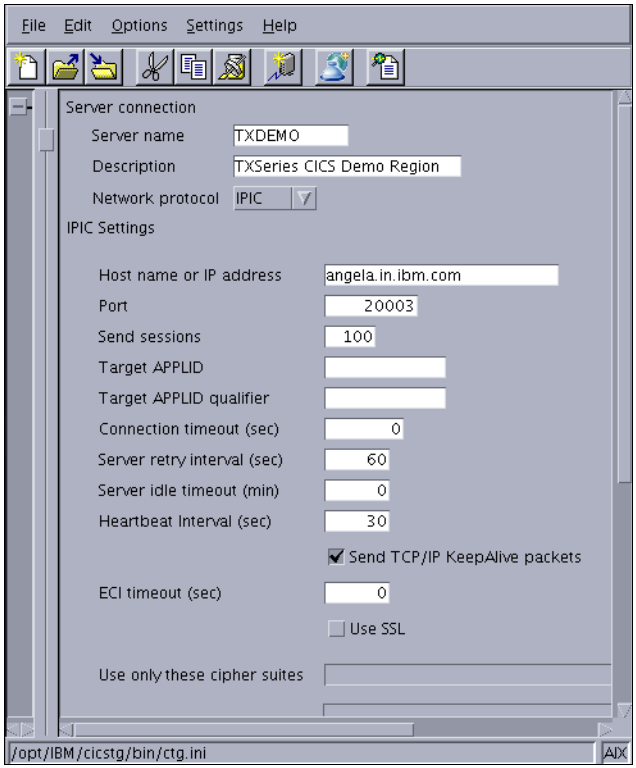


Figure 10-16 Define a new server

2. Define a new JSON web service in the **ctgcfg** tool, as shown in Figure 10-17 on page 451. The URI endpoint identifies the Uniform Resource Locator (URL) in which a JSON request is supposed to be sent. For example, in this case, it is the following URL:

   `http://<server> or <IP>:<CTG port>/hello`

   You need to complete the **Bind file** path details. This the same as the WSBIND file that was generated in the previous steps using the web service assistant tool.

3. The **Server name** to be completed in Figure 10-17 should map to the new server definition that we created in Figure 10-16 on page 450.

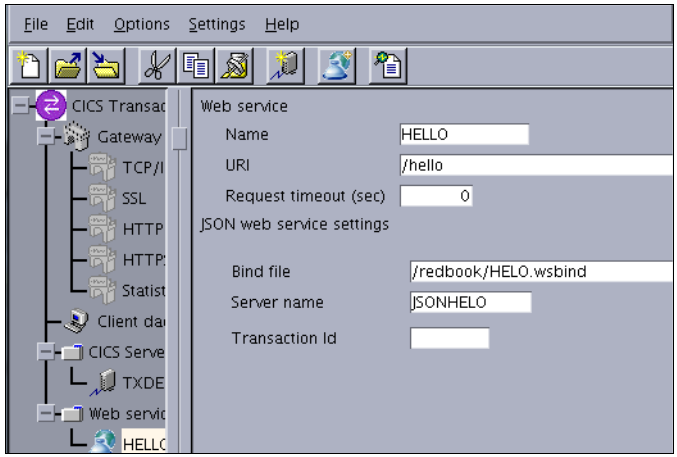


Figure 10-17 New JSON web service definition

4. Next, define an HTTP endpoint. Figure 10-18 shows how to create an HTTP endpoint using the **`ctgcfg`** tool. The port number defined here decides the port where the HTTP requests will be sent.

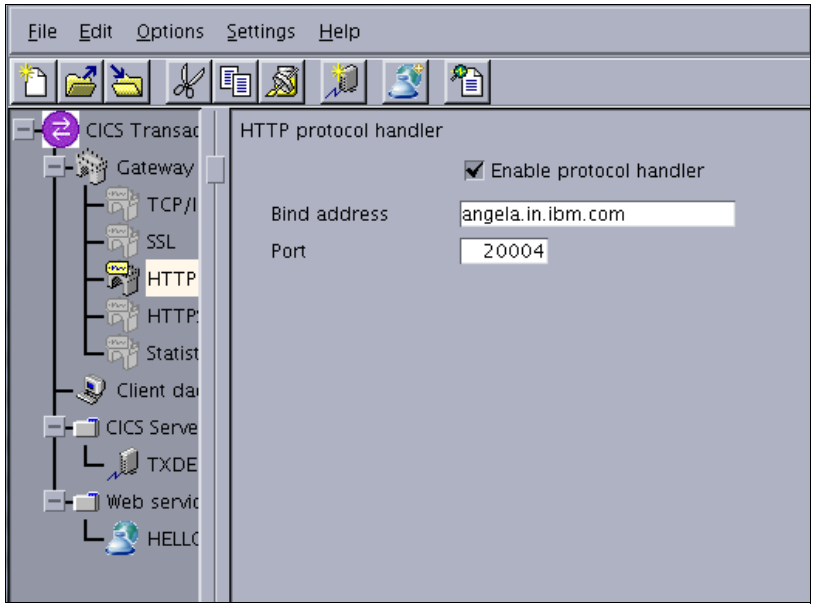


Figure 10-18 Creating a HTTP endpoint

5. This completes the CICS Transaction Gateway configuration. Save your changes by selecting **File** → **Save**.
6. We can now start the CICS TG using the **`ctgstart`** utility. This results in an HTTP listener in the port as defined in the configuration (Figure 10-19).

```
# ctgstart
ctgstart - CICS Transaction Gateway start program, Versi
on 9.1.0.0 Beta13 : Build Level c910-20140730-2155
(C) Copyright IBM Corporation 1996, 2014.  All rights re
served.
01/27/15 21:00:14:456 [0] CTG6400I CICS Transaction Gate
way is starting
01/27/15 21:00:14:530 [0] CTG8400I Using configuration f
ile /opt/IBM/cicstg/bin/ctg.ini
01/27/15 21:00:14:782 [0] CTG8461I Successfully initiali
zed trace plug-in 'com.ibm.ctg.client.FileTrace'
01/27/15 21:00:14:789 [0] CTG6577I Java details: Version
=1.7.0 - 32-bit, Path=/opt/IBM/cicstg/jvm170/bin/java
01/27/15 21:00:14:790 [0] CTG6574I Connection logging ha
s been disabled
01/27/15 21:00:15:022 [0] CTG6981I Successfully initiali
zed JNI library
01/27/15 21:00:15:394 [0] CTG6502I Initial Connection ma
nagers = 1, Maximum Connection managers = 100
01/27/15 21:00:15:396 [0] CTG6526I Initial workers = 1,
Maximum workers = 100
01/27/15 21:00:15:423 [0] CTG6505I Successfully created
the initial connection manager and Worker threads
01/27/15 21:00:15:460 [1] CTG8455I Successfully started
the local administration handler on port 2810
01/27/15 21:00:15:461 [0] CTG6597I The statistics API ha
ndler has not been started
01/27/15 21:00:15:542 [0] CTG6512I CICS Transaction Gate
way initialization complete
01/27/15 21:00:15:566 [0] CTG6411I Interval statistics a
re active with the statistics interval length set to 3 h
ours, 0 minutes and 0 seconds
01/27/15 21:00:15:567 [0] CTG6415I The statistics end of
 day time is set to 00:00:00 Greenwich Mean Time
CTG6508I To shut down the Gateway daemon type
CTG6493I    Q or - for normal shutdown
CTG6494I    I for immediate shutdown
```

Figure 10-19 The ctgstart command

This completes the configuration. The TXSeries program is now exposed as a JSON web service.

10.3.5 Testing the exposed TXSeries JSON web service

To test the JSON web service, we use a tool called **curl**. This is a freely available tool that sends JSON content to URLs.

Figure 10-20 on page 453 shows the sample output of the **curl** command.

```
C:\>curl -v -H "Content-Type: application/json" -X POST http://angela.in.ibm.com
:20004/hello --data {'HELLOOperation':{'hellofield':'Hello,TXSeries'}}
* About to connect() to angela.in.ibm.com port 20004 (#0)
*   Trying 9.121.255.19...
* connected
* Connected to angela.in.ibm.com (9.121.255.19) port 20004 (#0)
> POST /hello HTTP/1.1
> User-Agent: curl/7.27.0
> Host: angela.in.ibm.com:20004
> Accept: */*
> Content-Type: application/json
> Content-Length: 50
>
* upload completely sent off: 50 out of 50 bytes
< HTTP/1.1 200 OK
< X-Powered-By: Servlet/3.0
< Content-Type: application/json
< Content-Length: 57
< Date: Tue, 11 Nov 2014 20:58:59 GMT
<
{"HELLOOperationResponse":{"hellofield":"Hello, World!"}}* Connection #0 to host
 angela.in.ibm.com left intact
* Closing connection #0
```

Figure 10-20 curl command output

10.4 References

For more information about REST and JSON concepts, you can see the IBM Redbooks publication *Implementing IBM CICS JSON Web Services for Mobile Applications*, SG24-8161:

http://www.redbooks.ibm.com/redpieces/abstracts/sg248161.html?Open

You can see the IBM Knowledge Center for CICS Transaction Gateway V9.1:

http://www.ibm.com/support/knowledgecenter/SSZHFX_9.1.0/welcome.html

The json-schema-validator is an external tool that is useful to validate JSON content with the JSON schema:

http://json-schema-validator.herokuapp.com/

The JSONSchema is an external tool to generate a schema using JSON content:

http://www.jsonschema.net/

11

Migrating Oracle Tuxedo applications with IBM Migration Assistant

This chapter describes how to use the IBM Migration Assistant for Oracle Tuxedo (IBM Migration Assistant) to migrate Oracle Tuxedo applications to the IBM TXSeries environment.

This chapter provides information about the following topics:

11.1 IBM Migration Assistant for Oracle Tuxedo

IBM Migration Assistant for Oracle Tuxedo helps you migrate Oracle Tuxedo Common Business Oriented Language (COBOL) and C applications to a TXSeries environment.

The lightweight SupportPac can be installed over an existing TXSeries installation. It assists with Oracle Tuxedo application migrations by providing tools for the following phases of the process:

1. Analyzing the existing code
2. Building the new applications to work in a TXSeries environment
3. Deploying the new applications on TXSeries

11.1.1 Capabilities

IBM Migration Assistant can help you migrate client and server COBOL and C applications. The migrated applications then run seamlessly in a TXSeries region as part of IBM Customer Information Control System (IBM CICS) Transaction Server (CICS TS).

The migrated COBOL and C applications can map Tuxedo record types, error codes, and return values to their CICS counterparts, and vice versa. This capability enables seamless handling of record types and error codes in applications that were originally written by using Oracle Tuxedo's Application to Transaction Monitor Interface (ATMI) application programming interfaces (APIs).

11.1.2 Components

The IBM Migration Assistant for Oracle Tuxedo provides the following tools and libraries to aid in the different phases of application migration:

- Application scanner
- Build and infrastructure tools
- Glue And Stub generation
- IBM Migration Assistant library

Figure 11-1 on page 457 shows typical components of IBM Migration Assistant.

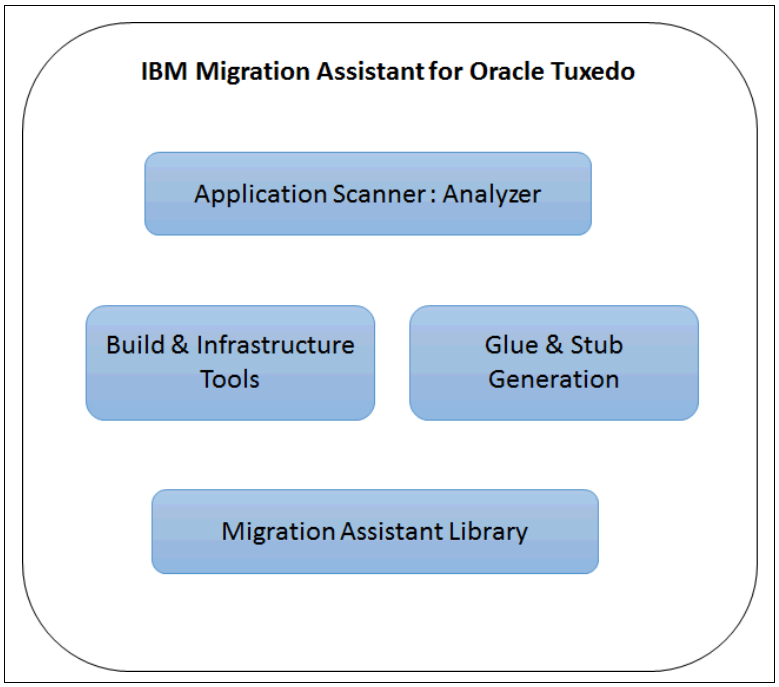


Figure 11-1 IBM Migration Assistant for Oracle Tuxedo

Application scanner

A code analyzer tool (available in the SupportPac) is advised for scanning existing Tuxedo applications to identify the Tuxedo ATMI APIs that were used, and whether they are supported in TXSeries. This ability makes the tool useful in identifying applications that can be readily migrated (with minimal or no code changes). Upon scanning the Tuxedo source base, the tool generates a report with details about each API and any modifications that are required for migration.

Build and Infrastructure tools

During the build phase, the Tuxedo source code of the customer is recompiled with the help of the IBM Migration Assistant-supplied build utilities, **`buildserver`** and **`buildclient`**. Because these utilities provide an interface that is similar to the interface of the Tuxedo utilities, it is not necessary to make any changes to the existing customer makefiles.

During the build process, the IBM Migration Assistant utilities compile the Tuxedo services for the TXSeries environment, linking them to the IBM Migration Assistant libraries. The build phase generates a loadable module, which can be loaded and run by a TXSeries application server.

Glue and stub generation

Glue and stub generation tools are used for internal processing to generate stubs and glue code for Tuxedo server applications. These tools are used by the **`buildserver`** tool to generate and link stub and glue code for Tuxedo server applications, so that the Tuxedo services can run in a CICS environment.

IBM Migration Assistant library

The SupportPac provides a set of libraries for migrating Oracle Tuxedo functionality to a TXSeries environment. The **`buildserver`** and **`buildclient`** tools map Tuxedo server client APIs to TXSeries APIs in the compile phase. In the link phase, the mapped APIs are linked to corresponding SupportPac libraries that act as an interface, which provides data compatibility between the two sets of APIs.

11.2 Advantages of using the IBM Migration Assistant for Oracle Tuxedo

IBM Migration Assistant for Oracle Tuxedo simplifies migration by automating all of the required changes for the new environment. Minimal or no changes to your existing applications or build infrastructure are required. This provides the following advantages.

11.2.1 Reduce costs

IBM TXSeries for Multiplatforms is distributed CICS online transaction processing (OLTP) software for mixed language applications. It provides excellent reliability, availability, and serviceability (RAS). The SupportPac adds value by providing a cost-effective, automated migration solution for Tuxedo-style COBOL and C applications. This reduces operational costs, and frees up your budget for newer business initiatives.

For more information about the Tuxedo-to-TXSeries migration process, see 11.3, “Four-step migration process” on page 459.

11.2.2 Future-proof investments to support business expansion

Seamless interoperability with CICS TS on IBM System z enables workloads to be moved easily, which creates room to accommodate future growth. TXSeries provides tooling and applications to support large computer network environments. Good availability of CICS skills and excellent quality of technical support enables smoother business expansion.

11.3 Four-step migration process

The process to migrate an Oracle Tuxedo application to a TXSeries environment can be completed in the following four steps:

1. Use code analysis to select suitable applications for migration.
2. Recompile and deploy the client applications on TXSeries.
3. Recompile and deploy the server applications on TXSeries.
4. Verify the deployed programs.

The migration process is designed to ensure that an application that is running in an Oracle Tuxedo-Oracle WebLogic environment can be deployed and then run in a TXSeries environment.

11.3.1 Analyze migration ready applications

It is important to select applications that are suitable for migration. This process includes analyzing the code to identify the applications that use the Tuxedo ATMI APIs that are supported for migration by the SupportPac. A code analyzer that is included in the SupportPac is an example of tooling that is available to assist with the identification task.

Example 11-1 shows a snippet from a sample Tuxedo ATMI application that is called.

Example 11-1 Tuxedo ATMI API usage in TUXCLT.c

```
....
if((Empld = (char *) tpalloc("STRING",NULL, sendlen))== NULL)
    {
            (void)fprintf(stderr,"Error allocating send buffer\n"):
               tpterm();
               Exit(1):
    }
....
```

The sample program uses the **tpalloc** and **tpterm** APIs. When the code analyzer scans the code, it detects the use of **tpalloc** and **tpterm** and marks it as SUPPORTED in the consolidated report that is generated, as shown in Example 11-2.

Example 11-2 Code analyzer report for TUXCLT.c

```
#######################################################################
# Consolidated Report #
#######################################################################
API NAME SUPPORTED UNSUPPORTED MODIFICATION NEEDED
tpalloc     1          0          0
tpterm      1          0          C
```

11.3.2 Build and deploy client applications

IBM Migration Assistant for Oracle Tuxedo provides a seamless build and deploy environment for client applications. A unified and flexible **buildclient** utility is provided to automate most of the client **buildprocess** tool. Similar interfaces to that of existing Tuxedo environment are available.

The **buildclient** tool builds client applications for the TXSeries environment by linking with migration assistant libraries. It also links the client application with the necessary IBM CICS Transaction Gateway libraries. The IBM Rational Application Developer tool is provided as a development platform to build and deploy applications.

Figure 11-2 shows the migration process for TUXEDO client applications.

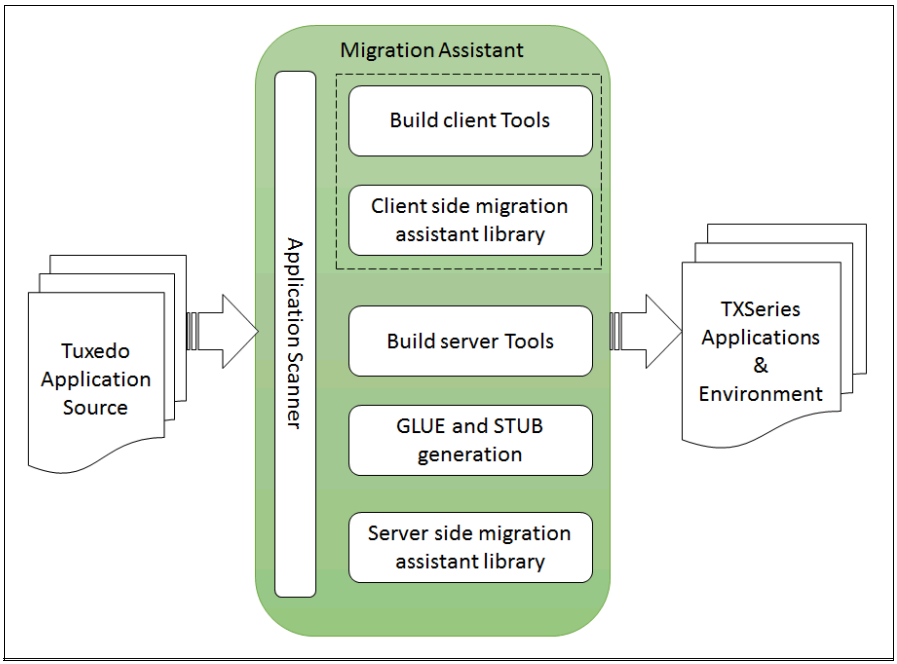


Figure 11-2 Migration process for TUXEDO client applications

Example 11-3 shows a snippet of the sample `TUXCLT.c` Tuxedo client application.

Example 11-3 Shows snippet of sample TUXCLT.c Tuxedo client application

```
 /* Tuxedo service DISPVAL */

void DISPVAL (TPSVCINFO *rqst)
{
   Char *Empld;
   Empld=(char *)rqst->data;
   empid=atol(Empld);

   userlog("DISPVAL: The Empld received is:(%s)", Empld;

   EXEC SQL declare dbcur cursor for
      select SALARY,DEPTCODE from EMP1 where EMPID=:empid;
   EXEC SQL open dbcur;
   EXEC SQL fetch dbcur into :sal, :deptcode;
```

```
   if (!sqlca.sqlcode)
   {
      userlog("DISPVAL:Successfully fetched record", sqlca.sqlcode);
   }
   else
   {
      userlog("DISPVAL:SQL Error:%d in fetching the record", sqlca.sqlcode);
      tpreturn(TPFAIL,4,NULL,0,0);
   }
}
```

To compile the client application, complete the following steps:

1. For the client to connect to the TXSeries region, you must configure TXSeries and CICS Transaction Gateway. In TXSeries, a listener must be configured to accept connection requests from CICS TG. Add a listener definition on the TXSeries region. Add a connection definition in the `CTG.INI` file of CICS TG server to connect to TXSeries region.
2. Before compiling and running the client, you must set a few environment variables in the `bldcltcfg.env` (`bldcltcfg.bat` for Windows) configuration file. Update the file by setting the required environment variables. Assuming `TX01` to be the CICS TG connection/server name added in step 1, the **`CICS_IMAT_CTG_SERVER_NAME`** environment variable must be set to `TX01`. This example assumes that the client is running in local mode.
3. Run the configuration file so that the variables are set in the shell where the client is run. This example assumes that the client application has the following code:

   ```
   makefile :
   all: client
   client: client.c
   buildclient -v -o TUXCLT -f
   clean:
   rm TUXCLT TUXCLT.o
   tuxclt.c
   ```

4. Compile the client by running the makefile. This runs the supplied **`buildclient`** command, which creates a client-executable program.

11.3.3 Build and deploy server applications

Tuxedo ATMI applications that are identified for migration during code analysis are compiled with the **`buildserver`** tool by using libraries from the SupportPac. The compiled modules can then be loaded onto a TXSeries region to be run.

Figure 11-3 shows the Tuxedo server application migration phase.

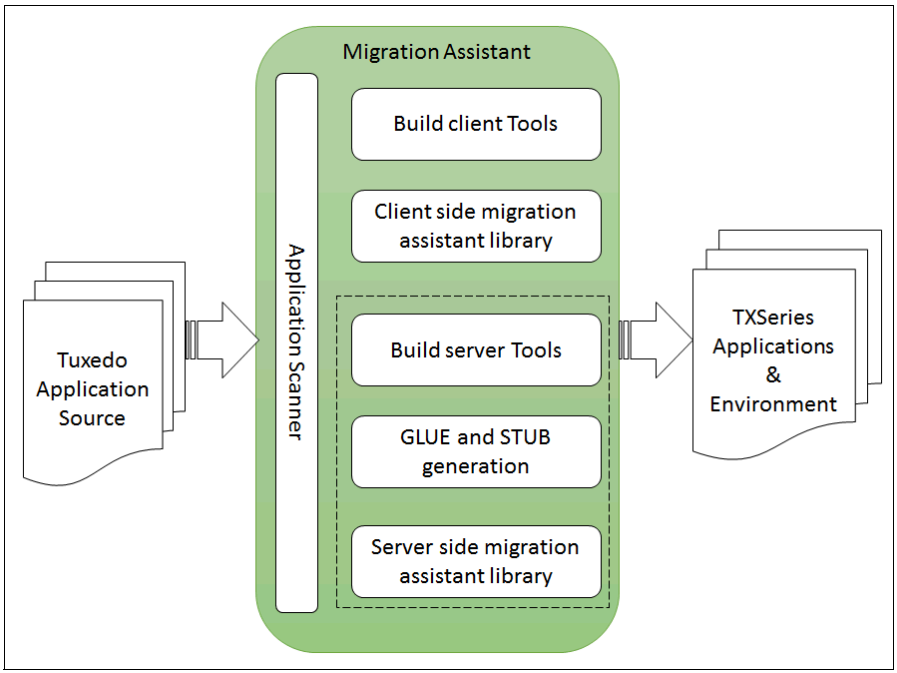


Figure 11-3 Tuxedo server application migration phase

Example 11-4 shows a snippet of the sample `TUXCLT.c` Tuxedo server application.

Example 11-4 shows snippet of sample TUXSVC.c Tuxedo server application

```
/* Tuxedo service DISPVAL */
void DISPVAL (TPSVCINFO *rqst)
{
char *Empld;
Empld=(char *)rqst->data;
empid=atol(Empld);
userlog("DISPVAL: The Empld received is:(%s)", Empld;
EXEC SQL declare dbcur cursor for
select SALARY,DEPTCODE from EMP1 where EMPID=:empid;
EXEC SQL open dbcur;
EXEC SQL fetch dbcur into :sal, :deptcode;
if (!sqlca.sqlcode)
{
userlog("DISPVAL:Successfully fetched record", sqlca.sqlcode);
}
```

```
else
{
userlog("DISPVAL:SQL Error:%d in fetching the record", sqlca.sqlcode);
tpreturn(TPFAIL,4,NULL,0,0);
}
userlog("DISPVAL:Values retrieved are: Salary:(%d) DeptCode:(%d)”,
sal,deptcode);
}
```

To compile the server application, complete the following steps:

1. Update the `bldsrvcfg.env` configuration file (`bldsrvcfg.bat` for Windows-based systems) to set the following environment variables:
   - Set the **CICS_IMAT_REGION_NAME** environment variable to the TXSeries region that should get created during the **buildserver** process. For example, **export CICS_IMAT_REGION_NAME=imat**.
   - Because the **TUXSVC** Tuxedo server program must also link to database libraries, you must set the **CICS_IMAT_DBLIBS** environment variable. For example, if using Oracle as the database, and linking to Oracle libraries, use the following command:

     ```
     export CICS_IMAT_DBLIBS="-L${ORACLE_HOME}/lib32 -lclntsh"
     ```
2. Run the configuration file so that the variables are set in the shell where the server is run. In this example, it is assumed that the following makefiles exist:

   ```
   all: services
   services: services.c
   buildserver -o TUXSVC -f services.c -s FETCHVAL -s UPDATEDB -s
   DISPVAL
   services.c: services.pc
   proc include=/usr/lpp/cics/include
   include=/usr/lpp/cics/supportpacks/im
   at/include auto_connect-yes \
   release_cursor=yes sqlcheck=syntax ireclen=512 iname=services.pc
   ```
3. Compile the server by running the previous makefile. This calls the buildserver utility supplied by IMAT, which creates the Tuxedo server program, **TUXSVC**. The **buildserver** utility also performs the other necessary actions required to run the services, such as creating a TXSeries region and adding the services to the TXSeries registry.

11.3.4 Verifying deployed programs

After the server programs are deployed in the TXSeries region, and the client programs are rebuilt and configured, the migration process is almost complete. The last step is to verify that the migration succeeded by using various log and trace files that are supplied with TXSeries and the IBM Migration Assistant for Oracle Tuxedo.

The following options are available to monitor the application and verify the successful migration:

- SupportPac server log

  You can monitor the execution of a migrated server program that is deployed in TXSeries by using the SupportPac server log.

  To enable the IMAT server-side log, set the **CICS_IMAT_SRV_LOG** environment variable to `1` in the environment file of the region, and then restart the region. After the IMAT server tracing is turned on, the trace messages are written to the `CSRVL.out` file, which is located under the `/var/cics_regions/regionname/data/` directory.

- Feature pack event log

  The IBM Migration Assistant provides an event log interface that is similar to Oracle Tuxedo's **USERLOG**. Developers use **USERLOG** to report events in the program, and the feature pack's event log can be enabled to do the same thing, as shown in the following section.

  To enable userlog tracing, set the **CICS_IMAT_USR_LOG** environment variable to `1` in the environment file of region, and then restart the region. After the userlog tracing is turned on, the trace messages are written to the `CULOG.out` file, which is located under the `/var/cics_regions/regionname/data/` directory. With this event log, users can include diagnostic messages to verify the successful execution of an application.

- SupportPac client log

  The IBM Migration Assistant provides internal tracing facilities at the client side.

  To enable IMAT tracing at the client side, set the **CICS_IMAT_CLT_TRACE** environment variable to `1` in the shell where the client is running. After trace is enabled, trace messages are written to the `CLTTRACE.pid.log` file.

11.3.5 The four-step checkpoint for migration

The four-step process for migrating Oracle Tuxedo applications is shown in Figure 11-4.

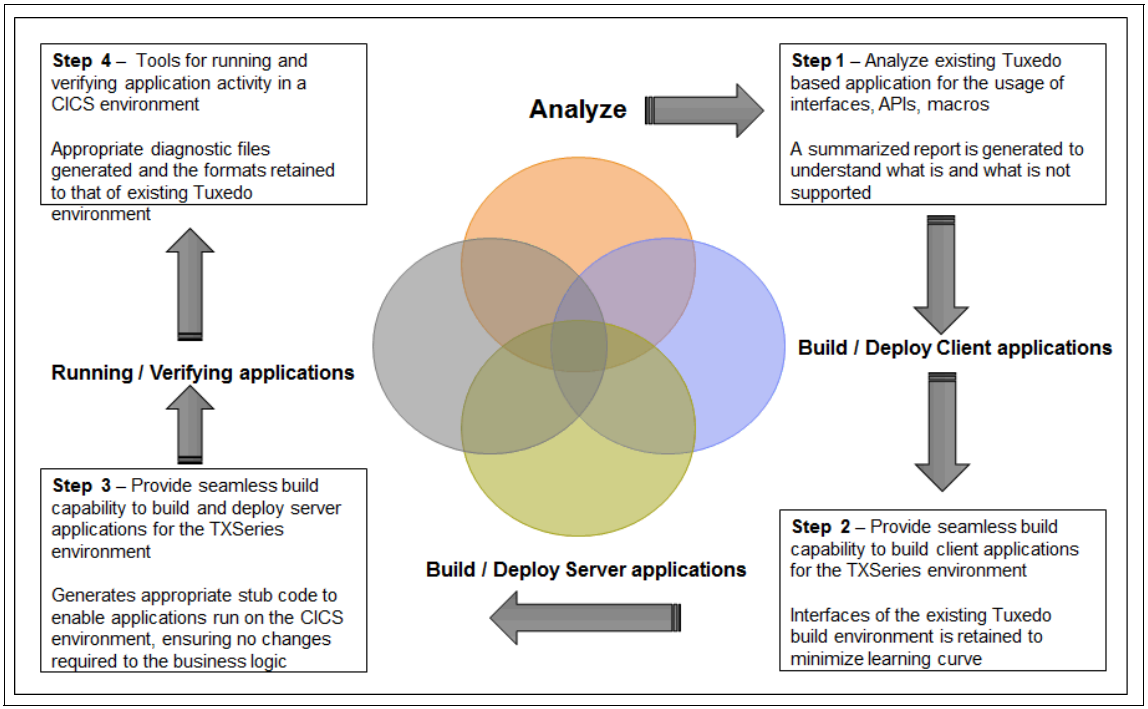


Figure 11-4 Four-step process for migrating Oracle Tuxedo applications

A

Systems Network Architecture: Configuration details

This appendix provides information about the configuration of the IBM Communications Server for AIX for connecting IBM TXSeries for Multiplatforms running on AIX to IBM Customer Information Control System (IBM CICS) Transaction Server on z/OS.

This appendix provides information about the following topics:

Systems Network Architecture (SNA) definition checklist

Table A-1 shows the relationship between different names and the actual values that we used to configure a working connection.

Table A-1 Names and values relationship

Key	IBM Virtual Telecommunications Access Method (IBM VTAM®)	CICS Transaction Server	TXSeries (stanza definition)	IBM Communications Server	Value
<1>	NETID		RemoteNetwork Name (CD)	Network Name	PSSCG3
<2>	Physical unit (PU)			Control Point	PONS1
<3>	Logical unit (LU)	NETNAME	LocalLU Name (RD)	LU	RAATLU42
<4>	Exchange identification (XID)			Node ID	05D05150
<5>	Token Ring Address			Remote Node MAC address	0006299cc4fb
<1>	NETID	Applid	RemoteLU Name (CD)	Partner LU	A6P0719L
<2>	Physical unit (PU)	Modename	DefaultSNA ModeName (CD)	Mode	CICSISC0

IBM Communication Server configuration file on AIX

To configure IBM Communication Server, we used the following steps:

1. Define the node.
2. Define the data link control (DLC).
3. Define ports.
4. Define link stations.
5. Define the LU6.2 local LU.
6. Define the LU6.2 partner LU.
7. Define the LU6.2 mode.

8. Define the partner LU6.2 location.
9. Define side information (optional).

Example A-1 shows the IBM Communications Server SNA configuration file that we defined for the BigBlueBank system to have a working connection from TXSeries for Multiplatforms running on AIX to CICS Transaction Server on z/OS.

Example: A-1 IBM Communication Server configuration file on AIX

```
[define_node_config_file]
major_version = 5
minor_version = 1
update_release = 1
revision_level = 189

[define_node]
cp_alias = pons1
description = ""
fqcp_name = PSSCG3.PONS1
node_type = NETWORK_NODE
mode_to_cos_map_supp = YES
mds_supported = YES
node_id = <00000000>
max_locates = 1500
dir_cache_size = 255
max_dir_entries = 0
locate_timeout = 0
reg_with_nn = YES
reg_with_cds = YES
mds_send_alert_q_size = 100
cos_cache_size = 24
tree_cache_size = 40
tree_cache_use_limit = 40
max_tdm_nodes = 0
max_tdm_tgs = 0
max_isr_sessions = 1000
isr_sessions_upper_threshold = 900
isr_sessions_lower_threshold = 800
isr_max_ru_size = 16384
isr_rcv_pac_window = 8
store_endpt_rscvs = NO
store_isr_rscvs = NO
store_dlur_rscvs = NO
cos_table_version = VERSION_1_COS_TABLES
send_term_self = NO
disable_branch_awareness = NO
```

```
cplu_syncpt_support = NO
cplu_attributes = NONE
dlur_support = YES
pu_conc_support = YES
nn_rar = 128
max_ls_exception_events = 0
ms_support = NORMAL
queue_nmvts = YES
clear_initial_topology = NO
ptf_flags = NONE

[define_ethernet_dlc]
dlc_name = ETHER0
description = ""
neg_ls_supp = YES
card_type = GDLC_ETHERNET
initially_active = NO
adapter_number = 0
max_saps = 16
ethernet_type = 802_3

[define_ethernet_port]
port_name = ETSAP0
description = ""
dlc_name = ETHER0
port_type = PORT_SATF
port_number = 0
lsap_address = 0x04
initially_active = YES
implicit_hpr_support = YES
implicit_link_lvl_error = NO
implicit_uplink_to_en = NO
max_rcv_btu_size = 1492
tot_link_act_lim = 255
inb_link_act_lim = 0
out_link_act_lim = 0
ls_role = LS_NEG
implicit_dspu_services = NONE
implicit_dspu_template = ""
implicit_ls_limit = 0
act_xid_exchange_limit = 9
nonact_xid_exchange_limit = 5
ls_xmit_rcv_cap = LS_TWS
max_ifrm_rcvd = 7
target_pacing_count = 7
```

```
max_send_btu_size = 1492
mac_address = <00096bbe175f>
implicit_cp_cp_sess_support = YES
implicit_limited_resource = NO
implicit_deact_timer = 30
effect_cap = 3993600
connect_cost = 0
byte_cost = 0
security = SEC_NONSECURE
prop_delay = PROP_DELAY_LAN
user_def_parm_1 = 128
user_def_parm_2 = 128
user_def_parm_3 = 128
local_name = ""
xid_timeout = 8
xid_retry_limit = 2
t1_timeout = 8
t1_retry_limit = 2
ack_time = 1
inact_time = 48
force_time = 120

[define_ethernet_ls]
ls_name = ETHL4
description = ""
port_name = ETSAP0
adj_cp_name = <00000000000000000000000000000000>
adj_cp_type = LEARN_NODE
dspu_services = NONE
dspu_name = <0000000000000000>
dlus_name = <00000000000000000000000000000000>
bkup_dlus_name = <00000000000000000000000000000000>
local_node_id = <00000000>
adj_node_id = <00000000>
mac_address = <0006299cc4fb>
lsap_address = 0x04
max_send_btu_size = 1492
ls_attributes = SNA
cp_cp_sess_support = YES
hpr_supported = NO
hpr_link_lvl_error = NO
auto_act_supp = NO
tg_number = 0
limited_resource = NO
solicit_sscp_sessions = NO
```

```
pu_name = <0000000000000000>
disable_remote_act = NO
default_nn_server = NO
link_deact_timer = 30
use_default_tg_chars = YES
effect_cap = 3993600
connect_cost = 0
byte_cost = 0
security = SEC_NONSECURE
prop_delay = PROP_DELAY_LAN
user_def_parm_1 = 128
user_def_parm_2 = 128
user_def_parm_3 = 128
target_pacing_count = 7
ls_role = USE_PORT_DEFAULTS
max_ifrm_rcvd = 0
dlus_retry_timeout = 0
dlus_retry_limit = 0
branch_link_type = NONE
adj_brnn_cp_support = ALLOWED
dddlu_offline_supported = NO
need_vrfy_fixup = NO
initially_active = YES
restart_on_normal_deact = NO
react_timer = 30
react_timer_retry = 65535
xid_timeout = 8
xid_retry_limit = 2
t1_timeout = 8
t1_retry_limit = 2
ack_time = 1
inact_time = 48
force_time = 120

[define_partner_lu]
plu_alias = A6PO719L
description = ""
fqplu_name = PSSCG3.A6PO719L
plu_un_name = A6PO719L
parallel_sess_supp = YES
appcip_routing_preference = NATIVE
max_mc_ll_send_size = 0
conv_security_ver = NO

[define_local_lu]
```

```
lu_alias = RAATLU42
list_name = ""
description = ""
lu_name = RAATLU42
lu_session_limit = 0
pu_name = <0000000000000000>
nau_address = 0
default_pool = NO
syncpt_support = NO
lu_attributes = NONE
sscp_id = 0
disable = NO
sys_name = ""
timeout = 60
back_level = NO

[define_mode]
mode_name = MDTCICS
description = ""
max_neg_sess_lim = 32767
plu_mode_session_limit = 2
min_conwin_src = 1
min_conloser_src = 0
auto_act = 0
receive_pacing_win = 4
max_receive_pacing_win = 0
default_ru_size = YES
max_ru_size_upp = 1024
max_ru_size_low = 0
cos_name = #CONNECT

[define_mode]
mode_name = CICSISCO
description = ""
max_neg_sess_lim = 32767
plu_mode_session_limit = 16
min_conwin_src = 8
min_conloser_src = 2
auto_act = 8
receive_pacing_win = 4
max_receive_pacing_win = 0
default_ru_size = YES
max_ru_size_upp = 1024
max_ru_size_low = 0
cos_name = #CONNECT
```

```
[define_tp]
tp_name = CICSTPN
description = ""
list_name = ""
conv_type = MAPPED
security_rqd = NO
sync_level = SYNCPT_NEGOTIABLE
enabled = YES
pip_allowed = YES
tp_instance_limit = 0
incoming_alloc_timeout = 0

[set_log_file]
log_file_type = AUDIT
action = NO_FILE_ACTION
file_name = /var/sna/sna.aud
backup_file_name = /var/sna/bak.aud
file_size = 1000000

[set_log_file]
log_file_type = ERROR
action = NO_FILE_ACTION
file_name = /var/sna/sna.err
backup_file_name = /var/sna/bak.err
file_size = 1000000

[define_trusted_groups]
group_id = 202
```

IBM Communications Server configuration file on Windows

To configure IBM Communications Server, we used the following steps:

1. Define the node.
2. Define a local area network (LAN) device.
3. Define a Common Programming Interface for Communications (CPI-C) and Advanced Program-to-Program Communication (APPC) peer connection.
4. Define a partner LU6.2 LU.
5. Define a local LU6.2 LU.
6. Define the mode.

Example A-2 shows the IBM Communications Server SNA configuration file that we defined for the BigBlueBank system to have a working connection from TXSeries for Multiplatforms running on Windows to CICS Transaction Server on z/OS.

Example: A-2 IBM Communication Server configuration files on Windows

```
NODE=(
     ANYNET_SUPPORT=NONE
     CP_ALIAS=PONS1
     DEFAULT_PREFERENCE=NATIVE
     DISCOVERY_SUPPORT=DISCOVERY_SERVER
     DLUR_SUPPORT=NORMAL
     FQ_CP_NAME=PSSCG3.PONS1
     MAX_LOCATES=150
     MAX_LS_EXCEPTION_EVENTS=200
     NODE_ID=05D05150
     NODE_TYPE=NETWORK_NODE
     REGISTER_WITH_CDS=1
     REGISTER_WITH_NN=NONE
     SEND_TERM_SELF=0
     TP_SECURITY_BEHAVIOR=VERIFY_EVEN_IF_NOT_DEFINED
)
PORT=(
     PORT_NAME=LAN0_04
     ACTIVATION_DELAY_TIMER=30
     ALLOW_ABM_XID_MISMATCH=0
     DELAY_APPLICATION_RETRIES=1
     DLC_DATA=00000000000004
     DLC_NAME=LAN
     IMPLICIT_BRANCH_EXTENDER_LINK=0
     IMPLICIT_CP_CP_SESS_SUPPORT=1
     IMPLICIT_DEACT_TIMER=0
     IMPLICIT_DSPU_SERVICES=NONE
     IMPLICIT_DSPU_TEMPLATE=00000000
     IMPLICIT_HPR_SUPPORT=1
     IMPLICIT_LIMITED_RESOURCE=NO
     IMPLICIT_LINK_LVL_ERROR=0
     LINK_STATION_ROLE=NEGOTIABLE
     MAX_ACTIVATION_ATTEMPTS=0
     MAX_IFRM_RCVD=7
     MAX_RCV_BTU_SIZE=65535
     PORT_TYPE=SATF
     RETRY_LINK_ON_DISCONNECT=1
     RETRY_LINK_ON_FAILED_START=1
```

```
        RETRY_LINK_ON_FAILURE=1
        DEFAULT_TG_CHARS=(
             COST_PER_BYTE=0
             COST_PER_CONNECT_TIME=0
             EFFECTIVE_CAPACITY=133
             PROPAGATION_DELAY=LAN
             SECURITY=NONSECURE
             USER_DEFINED_1=0
             USER_DEFINED_2=0
             USER_DEFINED_3=0
        )
        PORT_LAN_SPECIFIC_DATA=(
             ACK_DELAY=100
             ACK_TIMEOUT=10000
             ADAPTER_NUMBER=0
             BUSY_STATE_TIMEOUT=15
             IDLE_STATE_TIMEOUT=30
             INB_LINK_ACT_LIM=128
             LOCAL_SAP=04
             MAX_RETRY=10
             OUTSTANDING_TRANSMITS=16
             OUT_LINK_ACT_LIM=127
             POLL_TIMEOUT=8000
             POOL_SIZE=32
             REJECT_RESPONSE_TIMEOUT=10
             TEST_RETRY_INTERVAL=8
             TEST_RETRY_LIMIT=5
             TOT_LINK_ACT_LIM=255
             XID_RETRY_INTERVAL=8
             XID_RETRY_LIMIT=5
        )
   )
   LINK_STATION=(
        LS_NAME=LINK0001
        ACTIVATE_AT_STARTUP=1
        ACTIVATION_DELAY_TIMER=-1
        ADJACENT_BRANCH_EXTENDER_NODE=PROHIBITED
        ADJACENT_NODE_TYPE=LEARN
        AUTO_ACTIVATE_SUPPORT=0
        BRANCH_EXTENDER_LINK=1
        CP_CP_SESS_SUPPORT=0
        DEFAULT_NN_SERVER=0
        DELAY_APPLICATION_RETRIES=0
        DEPENDENT_LU_COMPRESSION=0
        DEPENDENT_LU_ENCRYPTION=OPTIONAL
```

```
     DEST_ADDRESS=0006299CC4FB04
     DISABLE_REMOTE_ACT=0
     DSPU_SERVICES=NONE
     HPR_LINK_LVL_ERROR=0
     HPR_SUPPORT=0
     INHERIT_PORT_RETRY_PARMS=1
     LIMITED_RESOURCE=NO
     LINK_DEACT_TIMER=0
     LINK_STATION_ROLE=USE_ADAPTER_DEFAULTS
     MAX_ACTIVATION_ATTEMPTS=-1
     MAX_IFRM_RCVD=0
     MAX_SEND_BTU_SIZE=65535
     NODE_ID=05D05150
     NULL_ADDRESS_MEANING=USE_WILDCARD
     PORT_NAME=LAN0_04
     RETRY_LINK_ON_DISCONNECT=0
     RETRY_LINK_ON_FAILED_START=0
     RETRY_LINK_ON_FAILURE=0
     REVERSE_ADDRESS_BYTES=0
     SOLICIT_SSCP_SESSION=0
     TG_NUMBER=0
     USE_DEFAULT_TG_CHARS=1
     USE_PU_NAME_IN_XID=0
     TG_CHARS=(
          COST_PER_BYTE=0
          COST_PER_CONNECT_TIME=0
          EFFECTIVE_CAPACITY=133
          PROPAGATION_DELAY=LAN
          SECURITY=NONSECURE
          USER_DEFINED_1=0
          USER_DEFINED_2=0
          USER_DEFINED_3=0
     )
)
DLUR_DEFAULTS=(
     DEFAULT_PU_NAME=PONS1
     DLUS_RETRY_LIMIT=3
     DLUS_RETRY_TIMEOUT=5
)
LOCAL_LU=(
     LU_NAME=RAATLU42
     DEFAULT_POOL=0
     LU_ALIAS=RAATLU42
     LU_SESSION_LIMIT=0
     NAU_ADDRESS=0
```

```
     ROUTE_TO_CLIENT=0
     SYNCPT_SUPPORT=0
)
MODE=(
     MODE_NAME=BLANK
     AUTO_ACT=0
     COMPRESSION=PROHIBITED
     COS_NAME=#CONNECT
     ENCRYPTION_SUPPORT=NONE
     DEFAULT_RU_SIZE=1
     MAX_INCOMING_COMPRESSION_LEVEL=NONE
     MAX_NEGOTIABLE_SESSION_LIMIT=8192
     MAX_OUTGOING_COMPRESSION_LEVEL=NONE
     MAX_RU_SIZE_UPPER_BOUND=1024
     MIN_CONWINNERS_SOURCE=4096
     PLU_MODE_SESSION_LIMIT=8192
     RECEIVE_PACING_WINDOW=3
)
MODE=(
     MODE_NAME=#BATCH
     AUTO_ACT=0
     COMPRESSION=PROHIBITED
     COS_NAME=#BATCH
     ENCRYPTION_SUPPORT=NONE
     DEFAULT_RU_SIZE=0
     MAX_INCOMING_COMPRESSION_LEVEL=NONE
     MAX_NEGOTIABLE_SESSION_LIMIT=8192
     MAX_OUTGOING_COMPRESSION_LEVEL=NONE
     MAX_RU_SIZE_UPPER_BOUND=2048
     MIN_CONWINNERS_SOURCE=4096
     PLU_MODE_SESSION_LIMIT=8192
     RECEIVE_PACING_WINDOW=20
)
MODE=(
     MODE_NAME=#BATCHC
     AUTO_ACT=0
     COMPRESSION=REQUESTED
     COS_NAME=#BATCH
     ENCRYPTION_SUPPORT=NONE
     DEFAULT_RU_SIZE=0
     MAX_INCOMING_COMPRESSION_LEVEL=LZ9
     MAX_NEGOTIABLE_SESSION_LIMIT=8192
     MAX_OUTGOING_COMPRESSION_LEVEL=LZ9
     MAX_RU_SIZE_UPPER_BOUND=2048
     MIN_CONWINNERS_SOURCE=4096
```

```
     PLU_MODE_SESSION_LIMIT=8192
     RECEIVE_PACING_WINDOW=20
)
MODE=(
     MODE_NAME=#BATCHCS
     AUTO_ACT=0
     COMPRESSION=REQUESTED
     COS_NAME=#BATCHSC
     ENCRYPTION_SUPPORT=NONE
     DEFAULT_RU_SIZE=1
     MAX_INCOMING_COMPRESSION_LEVEL=LZ9
     MAX_NEGOTIABLE_SESSION_LIMIT=8
     MAX_OUTGOING_COMPRESSION_LEVEL=LZ9
     MAX_RU_SIZE_UPPER_BOUND=2048
     MIN_CONWINNERS_SOURCE=4
     PLU_MODE_SESSION_LIMIT=8
     RECEIVE_PACING_WINDOW=3
)
MODE=(
     MODE_NAME=#BATCHSC
     AUTO_ACT=0
     COMPRESSION=PROHIBITED
     COS_NAME=#BATCHSC
     ENCRYPTION_SUPPORT=NONE
     DEFAULT_RU_SIZE=1
     MAX_INCOMING_COMPRESSION_LEVEL=NONE
     MAX_NEGOTIABLE_SESSION_LIMIT=8
     MAX_OUTGOING_COMPRESSION_LEVEL=NONE
     MAX_RU_SIZE_UPPER_BOUND=2048
     MIN_CONWINNERS_SOURCE=4
     PLU_MODE_SESSION_LIMIT=8
     RECEIVE_PACING_WINDOW=3
)
MODE=(
     MODE_NAME=#INTER
     AUTO_ACT=0
     COMPRESSION=PROHIBITED
     COS_NAME=#INTER
     ENCRYPTION_SUPPORT=NONE
     DEFAULT_RU_SIZE=1
     MAX_INCOMING_COMPRESSION_LEVEL=NONE
     MAX_NEGOTIABLE_SESSION_LIMIT=8192
     MAX_OUTGOING_COMPRESSION_LEVEL=NONE
     MAX_RU_SIZE_UPPER_BOUND=4096
     MIN_CONWINNERS_SOURCE=4096
```

```
     PLU_MODE_SESSION_LIMIT=8192
     RECEIVE_PACING_WINDOW=20
)
MODE=(
     MODE_NAME=#INTERC
     AUTO_ACT=0
     COMPRESSION=REQUESTED
     COS_NAME=#INTER
     ENCRYPTION_SUPPORT=NONE
     DEFAULT_RU_SIZE=1
     MAX_INCOMING_COMPRESSION_LEVEL=LZ9
     MAX_NEGOTIABLE_SESSION_LIMIT=8192
     MAX_OUTGOING_COMPRESSION_LEVEL=LZ9
     MAX_RU_SIZE_UPPER_BOUND=4096
     MIN_CONWINNERS_SOURCE=4096
     PLU_MODE_SESSION_LIMIT=8192
     RECEIVE_PACING_WINDOW=20
)
MODE=(
     MODE_NAME=#INTERCS
     AUTO_ACT=0
     COMPRESSION=REQUESTED
     COS_NAME=#INTERSC
     ENCRYPTION_SUPPORT=NONE
     DEFAULT_RU_SIZE=1
     MAX_INCOMING_COMPRESSION_LEVEL=LZ9
     MAX_NEGOTIABLE_SESSION_LIMIT=8
     MAX_OUTGOING_COMPRESSION_LEVEL=LZ9
     MAX_RU_SIZE_UPPER_BOUND=2048
     MIN_CONWINNERS_SOURCE=4
     PLU_MODE_SESSION_LIMIT=8
     RECEIVE_PACING_WINDOW=7
)
MODE=(
     MODE_NAME=#INTERSC
     AUTO_ACT=0
     COMPRESSION=PROHIBITED
     COS_NAME=#INTERSC
     ENCRYPTION_SUPPORT=NONE
     DEFAULT_RU_SIZE=1
     MAX_INCOMING_COMPRESSION_LEVEL=NONE
     MAX_NEGOTIABLE_SESSION_LIMIT=8
     MAX_OUTGOING_COMPRESSION_LEVEL=NONE
     MAX_RU_SIZE_UPPER_BOUND=2048
     MIN_CONWINNERS_SOURCE=4
```

```
     PLU_MODE_SESSION_LIMIT=8
     RECEIVE_PACING_WINDOW=7
)
MODE=(
     MODE_NAME=CICSISCO
     AUTO_ACT=8
     COMPRESSION=PROHIBITED
     COS_NAME=#CONNECT
     ENCRYPTION_SUPPORT=NONE
     DEFAULT_RU_SIZE=1
     MAX_INCOMING_COMPRESSION_LEVEL=NONE
     MAX_NEGOTIABLE_SESSION_LIMIT=128
     MAX_OUTGOING_COMPRESSION_LEVEL=NONE
     MAX_RU_SIZE_UPPER_BOUND=4096
     MIN_CONWINNERS_SOURCE=8
     PLU_MODE_SESSION_LIMIT=128
     RECEIVE_PACING_WINDOW=4
)
MODE=(
     MODE_NAME=QPCSUPP
     AUTO_ACT=0
     COMPRESSION=PROHIBITED
     COS_NAME=#CONNECT
     ENCRYPTION_SUPPORT=NONE
     DEFAULT_RU_SIZE=1
     MAX_INCOMING_COMPRESSION_LEVEL=NONE
     MAX_NEGOTIABLE_SESSION_LIMIT=1024
     MAX_OUTGOING_COMPRESSION_LEVEL=NONE
     MAX_RU_SIZE_UPPER_BOUND=1024
     MIN_CONWINNERS_SOURCE=512
     PLU_MODE_SESSION_LIMIT=1024
     RECEIVE_PACING_WINDOW=2
)
MODE=(
     MODE_NAME=QSERVER
     AUTO_ACT=0
     COMPRESSION=PROHIBITED
     COS_NAME=#CONNECT
     ENCRYPTION_SUPPORT=NONE
     DEFAULT_RU_SIZE=1
     MAX_INCOMING_COMPRESSION_LEVEL=NONE
     MAX_NEGOTIABLE_SESSION_LIMIT=64
     MAX_OUTGOING_COMPRESSION_LEVEL=NONE
     MAX_RU_SIZE_UPPER_BOUND=1024
     MIN_CONWINNERS_SOURCE=0
```

```
        PLU_MODE_SESSION_LIMIT=64
        RECEIVE_PACING_WINDOW=7
    )
    MODE=(
        MODE_NAME=SNASVCMG
        AUTO_ACT=0
        COMPRESSION=PROHIBITED
        COS_NAME=SNASVCMG
        ENCRYPTION_SUPPORT=NONE
        DEFAULT_RU_SIZE=0
        MAX_INCOMING_COMPRESSION_LEVEL=NONE
        MAX_NEGOTIABLE_SESSION_LIMIT=2
        MAX_OUTGOING_COMPRESSION_LEVEL=NONE
        MAX_RU_SIZE_UPPER_BOUND=512
        MIN_CONWINNERS_SOURCE=1
        PLU_MODE_SESSION_LIMIT=2
        RECEIVE_PACING_WINDOW=1
    )
    PARTNER_LU=(
        FQ_PLU_NAME=PSSCG3.A6PO719L
        ADJACENT_CP_NAME=PSSCG3.AFPYMOG3
        CONV_SECURITY_VERIFICATION=1
        MAX_MC_LL_SEND_SIZE=32767
        PARALLEL_SESSION_SUPPORT=1
        PARTNER_LU_ALIAS=A6PO719L
        PREFERENCE=USE_DEFAULT_PREFERENCE
    )
    TP=(
        TP_NAME=APINGD
        API_CLIENT_USE=0
        CONVERSATION_TYPE=EITHER
        DUPLEX_SUPPORT=EITHER_DUPLEX
        DYNAMIC_LOAD=1
        INCOMING_ALLOCATE_TIMEOUT=30
        LOAD_TYPE=0
        PATHNAME=C:\IBMCS\apingd.exe
        PIP_ALLOWED=1
        QUEUED=0
        RECEIVE_ALLOCATE_TIMEOUT=3600
        SECURITY_RQD=0
        SYNC_LEVEL=EITHER
        TP_INSTANCE_LIMIT=0
        TP_NAME_FORMAT=0
    )
    ADJACENT_NODE=(
```

```
     FQ_CP_NAME=PSSCG3.AFPYMOG3
     LU_ENTRY=(
          WILDCARD_LU=0
          FQ_LU_NAME=PSSCG3.A6PO719L
     )
)
SPLIT_STACK=(
     POOL_NAME=<None>
     STARTUP=1
)
SHARED_FOLDERS=(
     EXTENSION_LIST=(
     )
     CACHE_SIZE=256
)
LOAD_BALANCING=(
     ADVERTISE_FREQUENCY=1
     APPC_LU_LOAD_FACTOR=0
     DEFAULT_MAX_LU62_SESSIONS=512
     ENABLE_LOAD_BALANCING=0
     HOST_LU_LOAD_FACTOR=0
     LOAD_VARIANCE=3
)
VERIFY=(
     CFG_MODIFICATION_LEVEL=12
     CFG_VERSION_LEVEL=1
     CFG_LAST_SCENARIO=14
)
```

To know more details about configuring SNA using the **xsnaadmin** tool, see the following IBM developerWorks article on this topic:

http://www.ibm.com/developerworks/websphere/library/techarticles/0812_arpana/0812_arpana.html

B

Additional material

This IBM Redbooks publication refers to additional material that can be downloaded from the Internet as described in the following section.

Locating the web material

The web material associated with this IBM Redbooks publication is available in softcopy on the Internet from the IBM Redbooks web server:

ftp://www.redbooks.ibm.com/redbooks/SG247185

Alternatively, you can go to the IBM Redbooks publication website at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks publication form number, SG247185.

Using the web material

The additional web material that accompanies this IBM Redbooks publication includes the following files:

File name	*Description*
SG247185.zip	Compressed code samples as described and referred to throughout this book.

System requirements for downloading the web material

The following system configuration is suggested:

Hard disk space:	10 megabytes (MB) minimum
Operating System:	Windows
Processor:	500 megahertz (MHz) or higher
Memory:	256 MB, preferably 512 MB

How to use the web material

Create a subdirectory (folder) on your workstation, and extract the contents of the web material compressed file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed description of the topics covered in this IBM Redbooks publication.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 489. Note that some of the documents referenced here might be available in softcopy only:

- *Revealed! The Next Generation of Distributed CICS*, REDP-4099

Other publications

These publications are also relevant as further information sources:

- *Database Administration in task DB2 Knowledge center*
- *WebSphere MQ Security,* SC34-6588
- *TXSeries for Multiplatforms V8.1:*
  - *Installation Guide*, SC34-6632
  - *CICS Application Programming Guide*, SC34-6634
  - *CICS Problem Determination Guide*, SC34-6636
  - *CICS Administration Guide for Open Systems*, SC34-6637
  - *Application Programming Reference*, SC34-6640
  - *Administration Reference*, SC34-6641
  - *CICS Intercommunication Guide*, SC34-6644

Online resources

These websites and URLs are also relevant as further information sources:

- DW Answers

  https://developer.ibm.com/answers/
- TXSeries product support

  http://www.ibm.com/software/htp/txseries/support/
- TXSeries for Multiplatform documentation library (IBM Knowledge Center)

  http://www.ibm.com/support/knowledgecenter/SSAL2T/welcome
- XA and X/Open standards

  http://www.opengroup.org
- ACUCOBOL-GT and the other Acucorp tools

  http://www.acucorp.com/txseries
- Microsoft Help and Support

  http://support.microsoft.com/default.aspx?scid=kb;en-us;839013
- Java EE Connector Architecture

  http://java.sun.com/j2ee/connector
- CICS SupportPacs by Category

  http://www.ibm.com/software/htp/cics/support/supportpacs/category.html
- IBM Software - CICS Transaction Server for z/OS V3.1 - Library

  http://www.ibm.com/software/htp/cics/tserver/v31/library/
- IBM Software - CICS Transaction Gateway - Library

  http://www.ibm.com/software/htp/cics/ctg/library/
- IBM Software - WebSphere MQ - Library

  http://www.ibm.com/software/integration/wmq/library/

Stay connected with TXSeries

Stay tuned on our product page:

http://www.ibm.com/software/products/en/txseries/

Find us on Facebook:

https://www.facebook.com/pages/IBM-TXSeries-for-Multiplatforms/582838558411781?fref=ts

Follow us on Twitter:

https://twitter.com/txseries

Learn from us on developerWorks:

https://www.ibm.com/developerworks/community/blogs/DoMoreWithTXSeries/?lang=en

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications, and Additional materials, and order hardcopy Redbooks or CD-ROMs, at this website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

D

E

F

G

H

I

J

L

S

T

U

V

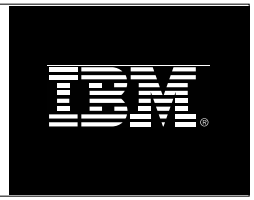
W

X

The Next Generation of Distributed IBM CICS

SG24-7185-01

ISBN 0738440574



(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

SG24-7185-01

ISBN 0738440574

Printed in U.S.A.