

Modernizing Applications with WebSphere eXtended Transaction Runtime

Improve productivity across
mixed-language applications

Optimize existing C and
COBOL assets

Manage CICS-style
data with DB2



John Kurian
Vivek M Anandaramu
Nageswararao V Gokavarapu
Toshiaki Saito
Madhavi Latha Sallam



International Technical Support Organization

**Modernizing Applications with WebSphere eXtended
Transaction Runtime**

December 2012

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

First Edition (December 2012)

This edition applies to WebSphere eXtended Transaction Runtime V2.1.

© Copyright International Business Machines Corporation 2012. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	v
Trademarks	vi
Preface	vii
The team who wrote this paper	vii
Now you can become a published author, too!	ix
Comments welcome	ix
Stay connected to IBM Redbooks	ix
Chapter 1. Introduction	1
1.1 Application infrastructure challenges	2
1.1.1 Impediments to enterprise growth	2
1.1.2 Business implications and needs	3
1.2 The WXTR solution	3
1.3 WXTR applicability	4
1.4 Customer benefits	5
1.5 High-level architecture	6
1.5.1 Tightly integrated, managed environment	6
1.5.2 Data management	7
1.5.3 Unified system administration	7
1.5.4 Integrated application development and debugging	7
Chapter 2. Feature highlights	9
2.1 Integration	10
2.1.1 Functional aspects	11
2.1.2 Synchronization of lifecycle activities	13
2.1.3 Deployment patterns	13
2.1.4 Global transaction support	18
2.1.5 Identity propagation	21
2.2 CICS data management support	22
2.2.1 VSAM file system by using DB2	23
2.2.2 Temporary storage queues	29
2.2.3 Transient data queues	29
2.3 Unified administration	30
2.3.1 Using the administrative console	30
2.3.2 Using the command line	32
2.3.3 Using a Jython script	35
2.4 Modernizing applications	36
2.4.1 Using Rational Developer for Power Systems Software	37
2.4.2 Debugging CICS COBOL applications	60
2.4.3 Creating binding classes	65
2.4.4 Accessing CICS COBOL through JCA and SCA	68
2.4.5 Creating an enterprise application archive	75
Chapter 3. Comparing WXTR with TXSeries for Multiplatforms	81
3.1 Introduction to TXSeries for Multiplatforms	82
3.2 Comparison of WXTR and TXSeries functionality	83
3.2.1 CICS API support	83
3.2.2 Migration from CICS servers	83

3.2.3	Integration with WebSphere Application Server	84
3.2.4	Supported software	84
3.2.5	System management functionality	85
3.2.6	Tooling for developers	85
3.3	Considerations when selecting WXTR versus TXSeries	86
3.3.1	Advantages of each approach	86
3.3.2	Key points to decide	87
3.4	Typical configurations for WXTR and TXSeries	88
3.4.1	WXTR configurations	88
3.4.2	TXSeries for Multiplatforms configurations	89
3.5	WXTR constraints	91
3.5.1	Configuration	91
3.5.2	Transactions	91
3.5.3	Application support	91
3.6	Conclusion	92
Chapter 4.	Migrating Java applications from Oracle WebLogic	93
4.1	Migration strategy and planning	94
4.1.1	Migration strategy considerations	94
4.1.2	Migration planning	94
4.2	Migration project plan	96
4.2.1	Migration implementation	96
4.3	Java EE application migration	97
4.4	Post deployment	98
4.5	WebSphere Application Server Migration Toolkit	98
4.6	Oracle Tuxedo Java client applications	99
4.7	Migration of WTC applications to WXTR	100
4.8	Conclusion	102
Chapter 5.	Modernizing Oracle Tuxedo applications with WXTR	103
5.1	WXTR Feature Pack for Modernizing Oracle Tuxedo Applications	104
5.1.1	Capabilities	104
5.1.2	Components	104
5.2	Advantages of using the WXTR modernization feature pack	106
5.2.1	Modernize Tuxedo COBOL and C applications	106
5.2.2	Reduce costs	106
5.2.3	WebSphere Application Server business capability	107
5.3	Using the WXTR modernization feature pack	107
5.4	Four-step migration process	107
5.4.1	Selecting migration-ready applications	107
5.4.2	Recompiling client and server applications	108
5.4.3	Deploying compiled client and server applications	111
5.4.4	Verifying deployed programs	112
5.4.5	Four-step checkpoint for migration	113
Related publications	115
IBM Redbooks	115
Online resources	115
Help from IBM	116

Notices

This information was developed for products and services offered in the U.S.A.

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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


Trademarks

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

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

AIX®
CICS®
DB2®
developerWorks®
IBM®

Informix®
Power Systems™
Power Systems Software™
Rational Team Concert™
Rational®

Redbooks®
Redbooks (logo) ®
System z®
WebSphere®
z/OS®

The following terms are trademarks of other companies:

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.

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

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

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

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

Preface

IBM® WebSphere eXtended Transaction Runtime V2.1 is an addition to the IBM Transaction Processing capabilities. This product provides a fast, scalable, and reliable transaction processing experience. Many customers have invested much time and effort in the development of business logic in CICS® style COBOL and C applications and are looking to unlock the value of those applications and extend them by using Java EE.

This paper helps you explore this product and provides information that helps you host your CICS style COBOL and C applications on a WebSphere platform. This paper also provides you with a detailed step-by-step approach for modernizing your existing Tuxedo-based applications through a migration to WXTR.

This paper is intended for developers and architects who want to extend and reuse their CICS style COBOL and C applications.

The team who wrote this paper

This paper was produced by a team of specialists from around the world working at India Software Labs, Bangalore, India.

John Kurian is an Advisory Software Engineer in IBM India Software Labs, Bangalore. He has eight years of experience in TXSeries for Multiplatforms and WebSphere eXtended Transaction Runtime fields. His professional expertise covers many technology areas that range from database, Java technologies, and CICS. He is working as a lead for WXTR development. He holds a Bachelors degree in Applied Electronics and Instrumentation from University of Kerala, India.

Vivek M Anandaramu is a Software Engineer in the WebSphere® organization at IBM India Software Labs. He has three years of experience in working with IBM TXSeries, WebSphere eXtended Transaction Runtime, IBM Migration Assistant for Oracle Tuxedo, and CICS products. He holds a degree in Computer Science from Visweswariah Technological University, Bangalore, India. He is a known entity in WebSphere eXtended Transaction Runtime blogging space. His areas of expertise include Java, J2EE programming, and Application Integration.

Nageswararao V Gokavarapu is a Software Engineer who has a Master of Engineering degree with Electronic Instrumentation as a specialization in Electronics and Communication Engineering department from Andhra University, Vishakhapatnam, India. He has been with IBM India software labs for the last six years. He has worked on distributed CICS transaction systems, Modernizing Applications with WebSphere eXtended Transaction Runtime, and TXSeries for Multiplatforms software. He is a prolific contributor to the IBM developer works community. His areas of interest include virtualization, transaction processing management, cloud computing, and UNIX systems.

Toshiaki Saito is a Software Engineer with IBM, Japan. He has four years of experience as a technical support engineer and Customer Support representative. He publishes technical documents on new features and products to Japan-based customers. His areas of expertise include CICS-TS, TXSeries for Multiplatforms, Modernizing Applications with WebSphere eXtended Transaction Runtime, CICS universal client, and CICS Transaction Gateway products.

Madhavi Latha Sallam is a Middleware specialist who works for IBM Software Labs, India. She has over six years of experience in IT industry. Her areas of expertise include WebSphere Application Infrastructure design, development, and management on System z® and distributed platforms, and Java/J2EE development projects. She is also part of the infrastructure practices team in India Software Lab Services and is involved in WebSphere infrastructure and performance projects. She holds a Bachelor of Engineering degree in Computer Science and Information Technology (CSIT) from J.N.T.U University Hyderabad, India.

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

- ▶ Amith N Kashyap, Developer
WebSphere eXtended Transaction Runtime
IBM Software Group, Bangalore, India
- ▶ Carla Sadtler, ITSO Project Leader
International Technical Support Organization, Raleigh Center
- ▶ Chris Rayns, ITSO Project Leader
International Technical Support Organization, Poughkeepsie Center
- ▶ Jeff Cox, ITSO Project Leader
International Technical Support Organization, Raleigh Center
- ▶ Sharad Kumar V. Deshpande, Marketing Manager
Enterprise Platform Software
IBM Software Group, Bangalore, India
- ▶ Matt Whitbourne, CICS Business Manager & Runtime PDT Lead
IBM Software Group, Hursley, UK
- ▶ Vijayeendra S. Namasevi, Development Manager
WebSphere eXtended Transaction Runtime & TXSeries
IBM Software Group, Bangalore, India
- ▶ Ashwini A. Deshpande, Developer
WebSphere eXtended Transaction Runtime
IBM Software Group, Bangalore, India
- ▶ Hariharan N. Venkitachalam, Development Lead
WebSphere eXtended Transaction Runtime
IBM Software Group, Bangalore, India

Now you can become a published author, too!

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

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

<http://www.ibm.com/redbooks/residencies.html>

Comments welcome

Your comments are important to us!

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

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

<http://www.ibm.com/redbooks>

- Send your comments in an email to:

redbooks@us.ibm.com

- Mail your comments to:

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

Stay connected to IBM Redbooks

- Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- Look for us on LinkedIn:

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

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

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

- Stay current on recent Redbooks publications with RSS Feeds:

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



Introduction

As business organizations grow, the experiences they gain and the strategies they pursue are built into computing applications that help them work more efficiently, reach more customers, and make more money. So, as technologies change, these organizations cannot abandon their existing applications, which represent significant financial investment and might still have good business value. The prudent approach, then, is to preserve and extend these business assets through incremental modernization.

IBM WebSphere eXtended Transaction Runtime (WXTR) is a distributed, online transaction processing environment for developing and hosting modern CICS COBOL applications. It is the latest in a series of IBM transaction processing tools that you use to progressively modernize your existing business applications to meet your needs.

This chapter introduces you to IBM WebSphere eXtended Transaction Runtime Version 2.1. The chapter includes the following sections:

- ▶ Application infrastructure challenges
- ▶ The WXTR solution
- ▶ WXTR applicability
- ▶ Customer benefits
- ▶ High-level architecture

1.1 Application infrastructure challenges

IT departments are continuously challenged to maintain their existing infrastructure and evolve that infrastructure to take advantage of the latest technologies. IBM recognizes that these efforts use significant resources and are developing solutions to help organizations effectively and efficiently scale their infrastructure to fit today's business needs.

1.1.1 Impediments to enterprise growth

Although enterprises seek expansion and diversification for growth, they face significant hurdles.

Operating environments are largely heterogeneous, consisting of business applications that are written in various programming languages and run on diverse platforms and systems. Often, these business applications operate in silos, each with its own development, maintenance, runtime, and administration environments, as shown in Figure 1-1.

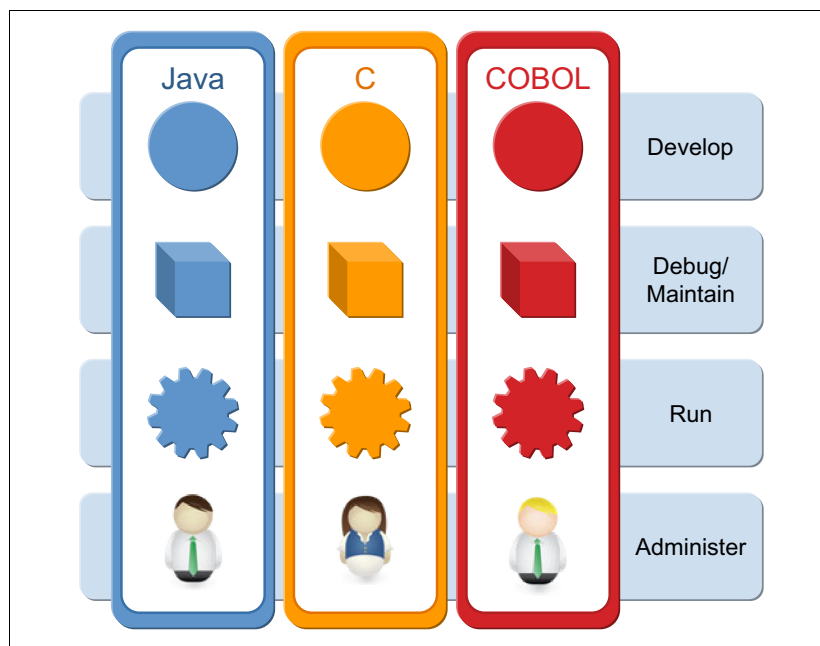


Figure 1-1 Organizational infrastructure challenges

Application architecture is mostly monolithic, which renders the entire infrastructure rigid and inflexible. This architecture limits the reuse and extension of existing, time-tested application business logic. Techniques exist to integrate applications, but these techniques can be elaborate and over time can become increasingly complex and expensive to maintain. Further complicating the issue are the evolving skill sets that are required to perform these types of complex integrations, which are not always available in the market.

1.1.2 Business implications and needs

These challenges, individually and collectively, limit the ability of a business to roll out new services and achieve accelerated time-to-market.

Even when integrations are possible, there is always concern about the adaptability and scalability of applications to support business expansion. Maintaining the various application environment silos also burdens IT budgets and reduces efficiency.

What is needed are new methods and techniques to simplify application infrastructure to improve operational efficiency and make the business more agile. Existing application assets and skills should be reused and extended to capitalize on existing investments and make resources available for new initiatives.

For maximum flexibility and speed, businesses need IT systems with the following features:

- ▶ A unified operational environment
- ▶ Adaptable, scalable infrastructure
- ▶ Interoperability and efficiency
- ▶ Skill sets that can be reused

1.2 The WXTR solution

IBM WebSphere eXtended Transaction Runtime (WXTR) was built with the needs that were described in “Business implications and needs” on page 3 in mind.

As an extension to WebSphere Application Server, WXTR provides a unified environment to host, manage, and maintain composite transactional applications that consist of COBOL, C, and Java components. Even composite applications with two phase commit transactions and mixed-language components can be easily deployed and maintained within WXTR and WebSphere Application Server. Security context, transaction context, and data can be seamlessly shared across the composite applications for improved architectural flexibility.

Deployment of WXTR, with WebSphere Application Server, provides enterprise-class scalability to composite applications. With WXTR and WebSphere Application Server as the core of a serving infrastructure, existing assets can be retained and extended in a modern, hybrid application environment, as shown in Figure 1-2 on page 4.

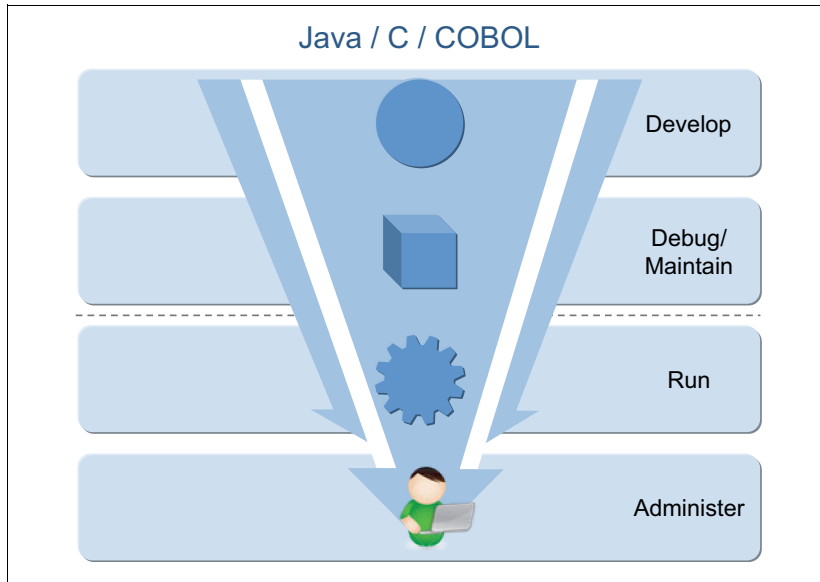


Figure 1-2 Hybrid application environment

1.3 WXTR applicability

WXTR is suitable for the following use cases that are comprised of application programs, together with their associated user interactions, and are called *workloads* (as shown in Figure 1-3 on page 5):

- Existing distributed COBOL and C workloads

WXTR allows customers to retain their existing COBOL and C assets as they consolidate the infrastructure around WebSphere Application Server.

The assets can be brought into an enterprise service-oriented architecture (SOA) environment by using the Java EE Connector Architecture (JCA) or service component architecture (SCA) interfaces in WXTR. These assets can then connect through enterprise service bus (ESB) to interoperate with workloads in WebSphere Process Server, WebSphere MQ, WebSphere Message Broker, or any other product that supports established MQ transport.

- Non-IBM transaction processing workloads (COBOL and C)

Customers that are currently running non-IBM transaction processing solutions, such as MicroFocus Enterprise Server or Clerity Unixix, can benefit from the use of WXTR. WXTR enables customers to modernize their COBOL and C assets as their future IT infrastructure is centered around WebSphere Application Server, which creates an agile, flexible, SOA-centric application infrastructure.

- Existing COBOL and C workloads that are running on IBM TXSeries

Existing TXSeries customers that have COBOL or C workloads that are extended into WebSphere Application Server can benefit from the tighter connectivity with Java workloads within WXTR. This configuration enables smoother and easier integration of COBOL and C assets into the SOA environment and improved management of mixed-language applications. The unified system management between WXTR and WebSphere Application Server makes it easy for administrators to deploy, administer, and manage COBOL and C assets within WebSphere Application Server, which results in improved efficiency.

- System z workloads that are moved to distributed platforms

CICS Transaction Server and WebSphere Application Server for z/OS® are the premium hosting environments for modern COBOL and C applications that are extended into Java EE. For customers that do not require the benefits of the System z platform and want to continue investing in application modernization, WXTR provides a distributed alternative.

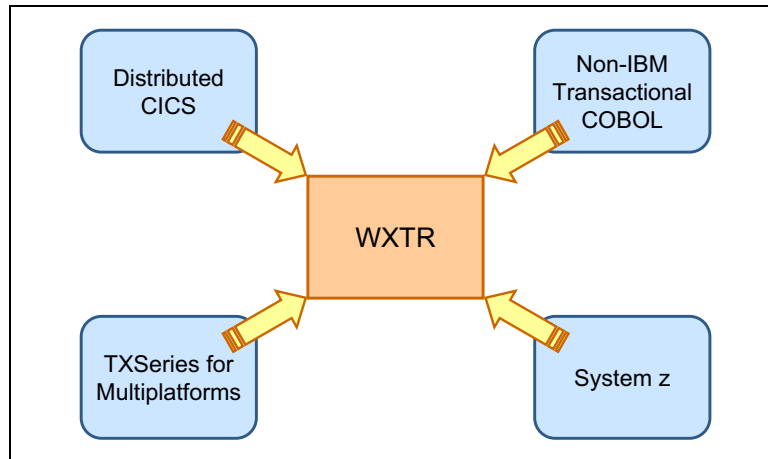


Figure 1-3 Types of WXTR workloads

1.4 Customer benefits

WXTR provides a broad range of benefits that result in the following advantages:

- Creates highly responsive, tightly integrated application-serving infrastructures for COBOL, C, and Java workloads. This integrated infrastructure enables the optimization, deployment, and extended reuse of existing COBOL or C assets as services across a range of platforms within an SOA-environment.
- Capitalizes on enterprise-wide skill sets to effectively optimize and modernize COBOL and C assets and consolidates the infrastructure around WebSphere Application Server.
- Increases operational efficiency by using the simplified, unified administration capability that is enabled by WXTR to centrally manage COBOL, C, and Java workloads.
- Simplifies integration complexity and improves the agility of any SOA-enabled application infrastructure to support growth.
- Extends existing transactional COBOL and C business applications into WebSphere Application Server to use the WebSphere portfolio's extensive business capabilities on distributed platforms.
- Improves productivity and efficiency by using enhancements that enable seamless problem determination and resolution across mixed-language applications.

1.5 High-level architecture

A high-level view of the components and architecture that are related to WXTR is shown in Figure 1-4.

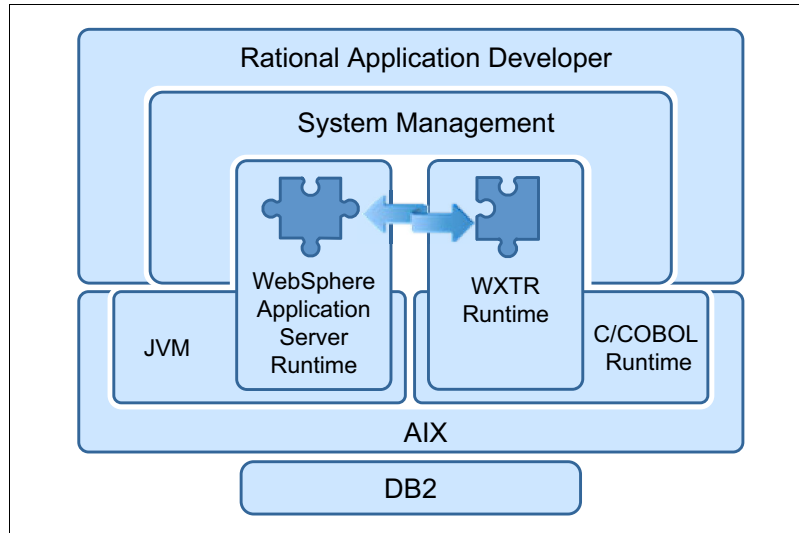


Figure 1-4 High-level architecture of WXTR

The two major components of the architecture, the run times for WebSphere Application Server and WXTR, exist separately but are joined with an adapter. The WebSphere run time runs on a Java virtual machine (JVM), whereas the WXTR run time is a collection of native operating system processes. The WXTR run time runs COBOL and C applications and the WebSphere run time runs Java EE applications.

Both run times run on the AIX® platform. The WXTR run time must connect to a DB2® or Oracle database, which can be found in the same machine as WXTR or in a remote machine.

The different functions that WXTR supports are classified into the following categories:

- ▶ Tightly integrated, managed environment
- ▶ Data management
- ▶ Unified system administration
- ▶ Integrated application development and debugging

1.5.1 Tightly integrated, managed environment

WXTR includes functions that together provide a tightly integrated, managed environment to host Java, C, and COBOL applications that are under the same WebSphere Application Server instance.

A Java application that is found within WebSphere Application Server can access a C or COBOL application in WXTR by using the JCA or SCA interfaces. Application errors that arise within the C or COBOL executable file are propagated back to the Java application as Java exceptions and handled accordingly.

User data is passed among Java, C, and COBOL applications by using COMMAREA in a character format. Popular integrated development environments such as Rational® Application Developer™ provide a binding feature to simplify this mapping process. For more information, see 2.1, “Integration” on page 10.

1.5.2 Data management

WXTR provides support for CICS data management facilities, such as files and queues that are stored in a DB2 database. WXTR supports record-level locking on fixed-length and variable-length records, and allows VSAM-style access to the data stored in DB2.

WXTR also supports temporary storage and transient data queues. The temporary storage queues are used for shared reading, writing, and updating of data by multiple transactions. The transient data queue service provides a generalized queuing facility for internal or external processing. Selected data that are specified in the application program can be routed to or from predefined symbolic destinations, called *extra-partitions*. These extra-partition destinations are queues that are found on any system resource and can be accessed by programs within and across regions. Any system file can be processed sequentially with WXTR transient data.

For more information about data management functions, see 2.2, “CICS data management support” on page 22.

1.5.3 Unified system administration

WXTR works in a two-in-one mode by using a highly optimized local adapter, with the WebSphere run time handling the Java EE applications and the WXTR run time handling the COBOL and C applications. But this configuration does not result in more administrative overhead because the COBOL and Java EE assets can be managed within the WebSphere Application Server administrative console. This management ability reduces the learning curve for administrative personnel and allows them to easily and efficiently deploy, administer, and optimize services. It also allows for the creation of user groups with predefined privileges to run certain administrative tasks, thus reducing the dependence on administrative personnel and improving overall productivity.

Administrators can view and modify the WXTR service properties through the WXTR administration panel in the WebSphere administrative console or by using WebSphere administration scripts. WXTR also enables non-root users to complete a predefined set of administrative tasks without the need for root user privileges.

For more information about WXTR administration functions, see 2.3, “Unified administration” on page 30.

1.5.4 Integrated application development and debugging

Rational Developer for Power Systems, together with WXTR, can provide an easier and more productive application development experience across Java EE and COBOL applications. You can develop, deploy, and debug applications within a single integrated development environment to improve productivity.

Application developers can step into and inspect a COBOL program from a Java EE application. Similarly, data exchange between COBOL and Java EE applications can be achieved by using the CICS/IMS Java Data Binding feature of Rational Application Developer.

For more information about the use of the IBM Rational tools, see 2.4.1, “Using Rational Developer for Power Systems Software” on page 37.



Feature highlights

This chapter describes the features and interfaces that are provided by IBM WebSphere eXtended Transaction Runtime (WXTR) to modernize existing enterprise applications, such as Customer Information Control Systems (CICS). It also explains the WXTR mechanism for passing application data between Java and COBOL or C applications, and error propagation and exception handling.

This chapter includes the following sections:

- ▶ Integration
- ▶ CICS data management support
- ▶ Unified administration
- ▶ Modernizing applications

2.1 Integration

WXTR provides an execution environment for hosting COBOL and C business applications within IBM WebSphere Application Server. It offers a tightly integrated and managed environment for hosting Java Enterprise Edition (Java EE) and modern C or COBOL applications with interoperability between the workloads of each application.

WXTR provides native runtime connectivity with standard interfaces between Java EE and COBOL applications, which significantly eases integration in an enterprise environment. Java applications can connect and start C and COBOL applications through standard connection interfaces that are based on Java EE Connector Architecture (JCA) or Service Component Architecture (SCA).

WXTR also offers a unique system management capability that is tightly integrated with WebSphere Application Server, as shown in Figure 2-1. This capability enables easy deployment, administration, and optimization of services, and it significantly increases efficiency for managing mixed-language transactional applications.

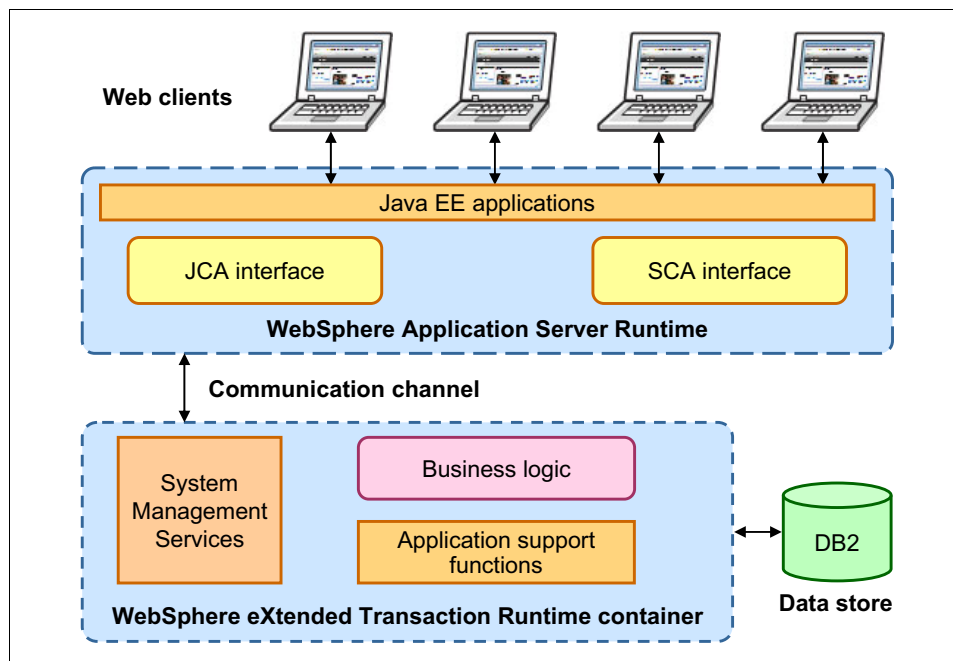


Figure 2-1 WXTR is a tightly integrated, managed environment for Java and COBOL applications

WXTR provides a buffer to pass application data from Java applications to C and COBOL applications. It also propagates the C and COBOL Runtime System (RTS) and sends application errors as Java exceptions back to the Java applications for further processing.

Thanks to these and other features, WXTR enables application developers to focus on business logic rather than on getting Java EE and COBOL assets to interoperate with sufficient failure detection, recovery, and access to data.

2.1.1 Functional aspects

With WXTR, Java application developers can use the standard JCA and SCA interfaces to invoke COBOL applications. This section provides an introduction to the JCA and SCA interfaces, the mechanism for passing data from Java to C or COBOL programs, error propagation, and exception handling.

JCA interface

Today's e-business environment may demand that companies upgrade their application infrastructure and modernize their existing assets. In doing so, they will likely need to integrate the business logic and historical data available in existing enterprise applications, such as Customer Information Control Systems (CICS).

Frequently, critical business transactions were originally written in procedural languages, such as COBOL or C. WXTR offers a Java EE platform-specified, JCA-standard interface that developers can use to access Java EE applications and data, as shown in Figure 2-2.

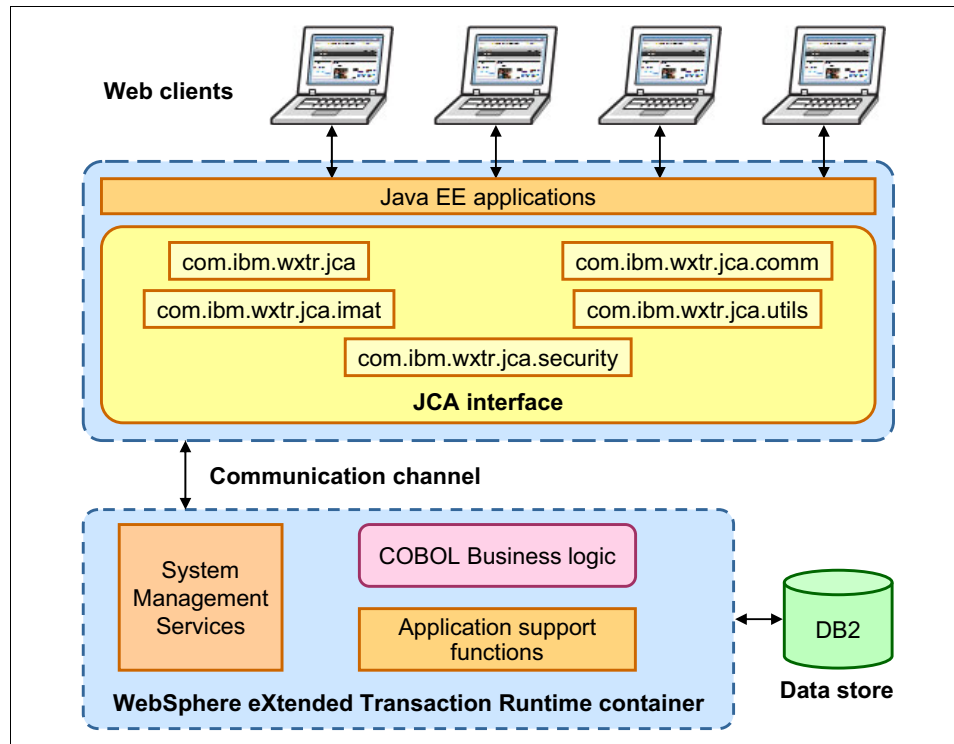


Figure 2-2 JCA interface for Java applications

WXTR provides JCA Common Client Interface (CCI) APIs to interact with COBOL programs and data from Java EE programs. Java application developers can use the WXTR-supplied `com.ibm.wxtr.jca` and `com.ibm.wxtr.jca.utils` packages to create connections to and interact with CICS C or COBOL applications.

SCA interface

SCA provides a model for creating applications that follow service-oriented architecture (SOA) principles. WXTR supports the SCA programming model by using the SCA Feature Pack for WebSphere Application Server.

WXTR supplies an SCA service that is called *WXTRService* and an SCA component that is called *WXTRServiceComponent*. The invocation of a COBOL program that is deployed in WXTR is shown in Figure 2-3.

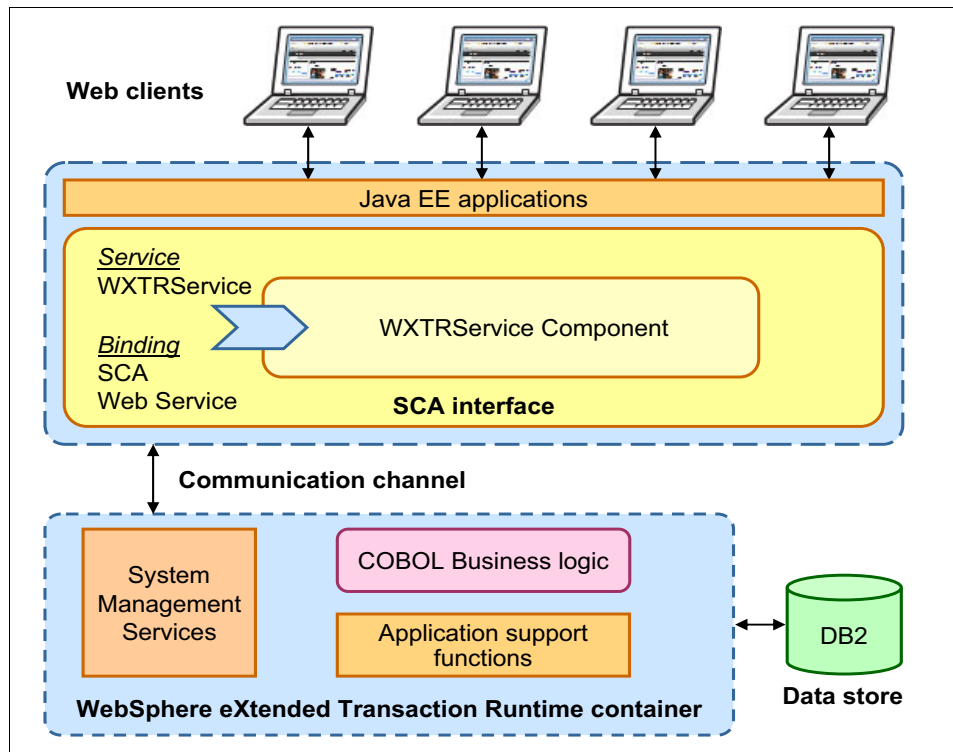


Figure 2-3 SCA interface for Java applications

The SCA service is bound to the default SCA binding and web service binding. Because of this configuration, any SCA application or web service client application can use the SCA service to start COBOL applications that are deployed in WXTR.

Error handling

WXTR provides a unique error handling mechanism that makes it possible to receive COBOL errors as Java exceptions. When a WXTR COBOL application is invoked by a Java client through the JCA or SCA interfaces and exits abnormally because of an abend, this information is relayed to the client.

For a JCA client, a `ResourceException` is thrown by the `execute` method of the `WXTRInteraction` object. This exception provides information about the cause, the error message, and so on. For an SCA client, a `COBOLExecException` is thrown by the methods of the `WXTRService` service that is provided by the `WXTRServiceComponent`. The `COBOLExecException` provides details, such as the error message and fault information.

For more information about the error propagation procedures for the JCA and SCA interfaces, see “Error propagation” on page 73.

Application data support

WXTR provides a data buffer that is called COMMAREA that is used for sending application data from Java applications to C/COBOL applications. This data buffer supports application data up to 32 KB. The WXTR-supplied `com.ibm.wxtr.jca.utils` package provides two Java classes (`JavaStringRecord` and `JavaBytesRecord`) that use a streamable interface to create data buffers or records that are to be sent to C and COBOL programs. WXTR also supplies code snippets for passing application data in character format. If your program must send mixed data types, specify the `JavaBytesRecord` class instead of the `JavaStringRecord` class.

By using the CICS/IMS Java Data Binding feature that is provided with Rational Application Developer for WebSphere, developers can share mixed-type data across Java EE and COBOL environments. The Java data binding tool includes COBOL Importer, with which you import existing COBOL programs and generates metadata information about the COBOL data structures. Through a process of datatype transformation, COBOL Importer maps the data types in the source file to corresponding data types that can be accessed in a Java program, which provides a convenient way to transfer data between Java EE and COBOL applications.

A Java application starts a CICS COBOL program with COMMAREA data by creating a `WXTRInteraction` object through the `createInteraction()` method of `WXTRConnection` class and invoking the `execute()` method of `WXTRInteraction` class. For more information, see WebSphere eXtended Transaction Runtime Information Center at this website:

<http://pic.dhe.ibm.com/infocenter/wxtrform/v2r1/index.jsp>

2.1.2 Synchronization of lifecycle activities

The term *lifecycle* refers to the states that an object, such as a container or an application server, goes through from the time it is started until the time it is stopped. *Lifecycle activities* refers to the operations that transfer the object from one state to another. For a container or an application server, these lifecycle activities include operations such as start, stop, and restart.

When you consider a topology that consists of Java EE and CICS applications that are deployed in separate products, you must manage their lifecycle activities separately. Because each product features its own startup modes, administration utilities, and so on, it is difficult to coordinate, synchronize, and automate the operation of both products.

In contrast, because a WXTR container exists as another container within WebSphere Application Server, its lifecycle activities are synchronized with those activities of WebSphere Application Server. A WXTR container is a named collection of resources that are controlled by WXTR as a unit. For example, when WebSphere Application Server is started, the WXTR container also is started. When WebSphere Application Server is shut down, the WXTR container also is shut down.

2.1.3 Deployment patterns

WXTR can be deployed in various ways to meet business and user expectations. Because WXTR exists as a container within WebSphere Application Server, its deployment patterns depend on how WebSphere Application Server is deployed.

WebSphere Application Server supports various kinds of deployments, from a single, stand-alone server to more complex deployments that consist of clusters of application servers that are spread across physical systems. Each deployment is an individual WebSphere Application Server runtime environment, also known as a *profile*.

A profile is a runtime instance of WebSphere Application Server that consists of specific components. You can use WebSphere Application Server profile templates that contain predefined components to create your profiles. For example, if you need only one stand-alone application server, you can create it by using the default profile template. If you want to create a profile to act as an endpoint in a distributed deployment (on which servers can be created later on), you can use a custom template.

Important: WXTR Version 2.1 is supported by WebSphere Application Server Base, WebSphere Application Server Express, WebSphere Application Server Developer Edition, and WebSphere Application Server Network Deployment. Each version supports different profile types, but describing them all is outside of the scope of this Redpaper publication. For more information, see the WebSphere Application Server Information Center at this website:

[.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.installation.express.doc/info/exp/ae/cpro_overview.html](http://www.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.installation.express.doc/info/exp/ae/cpro_overview.html)

WXTR V2.1 supports the following profile types that WebSphere Application Server v7.1 and v8.0 provides:

- ▶ **Default:** A default profile is used to create single stand-alone WebSphere Application Server instances that are configured with WXTR.
- ▶ **Dmgr:** A dmgr profile consists of WebSphere Application Server deployment manager that acts as the central administration unit for administering, configuring, and monitoring all WXTR instances.
- ▶ **Custom:** A custom profile acts as an endpoint in a distributed topology. You can create a custom profile to serve as an endpoint, add it to your deployment manager, and then use the deployment manager to create a server or a cluster member on the endpoint.

Any WXTR topology can be created by using these profile types as building blocks, from a testing and development space to a full production environment. Details about these two topologies are provided in the following sections.

Topologies for testing or development

Testing and development deployments often need a single individual server or group of individual servers, with the group organized into a cluster.

Single individual server

A WXTR deployment with a single individual (or stand-alone) WebSphere Application Server (as shown in Figure 2-4) can be created by using any WebSphere Application Server product that is supported by WXTR.

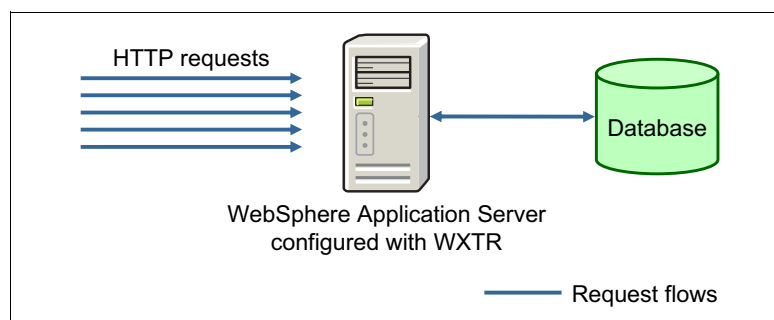


Figure 2-4 Single individual (or stand-alone) server deployment

Group of individual servers

You can create a group of individual application servers and manage them individually, or you can group them under a single deployment manager and manage them together (as shown in Figure 2-5). Although managing individual servers is supported by any WXTR-compatible WebSphere Application Server product, managing a group of servers under one deployment manager requires WebSphere Application Server Network Deployment.

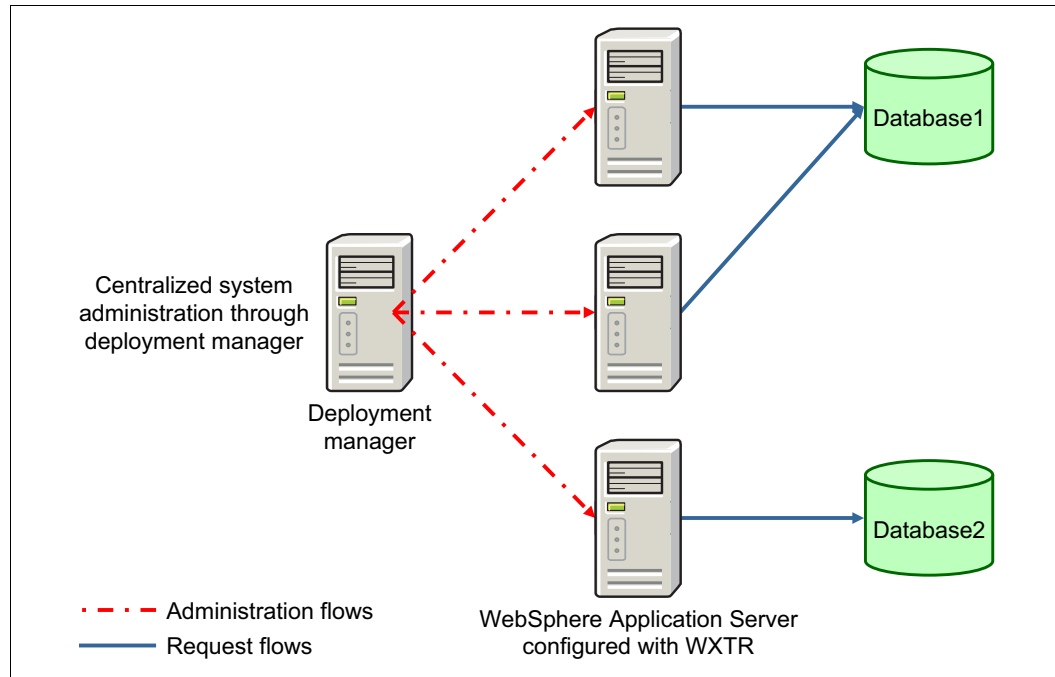


Figure 2-5 Group deployment of individual servers under one deployment manager

Small cluster of application servers

Because production systems often consist of clustered application servers, you might want to create a similarly clustered environment for testing purposes (see Figure 2-6 on page 16). This arrangement also requires WebSphere Application Server Network Deployment.

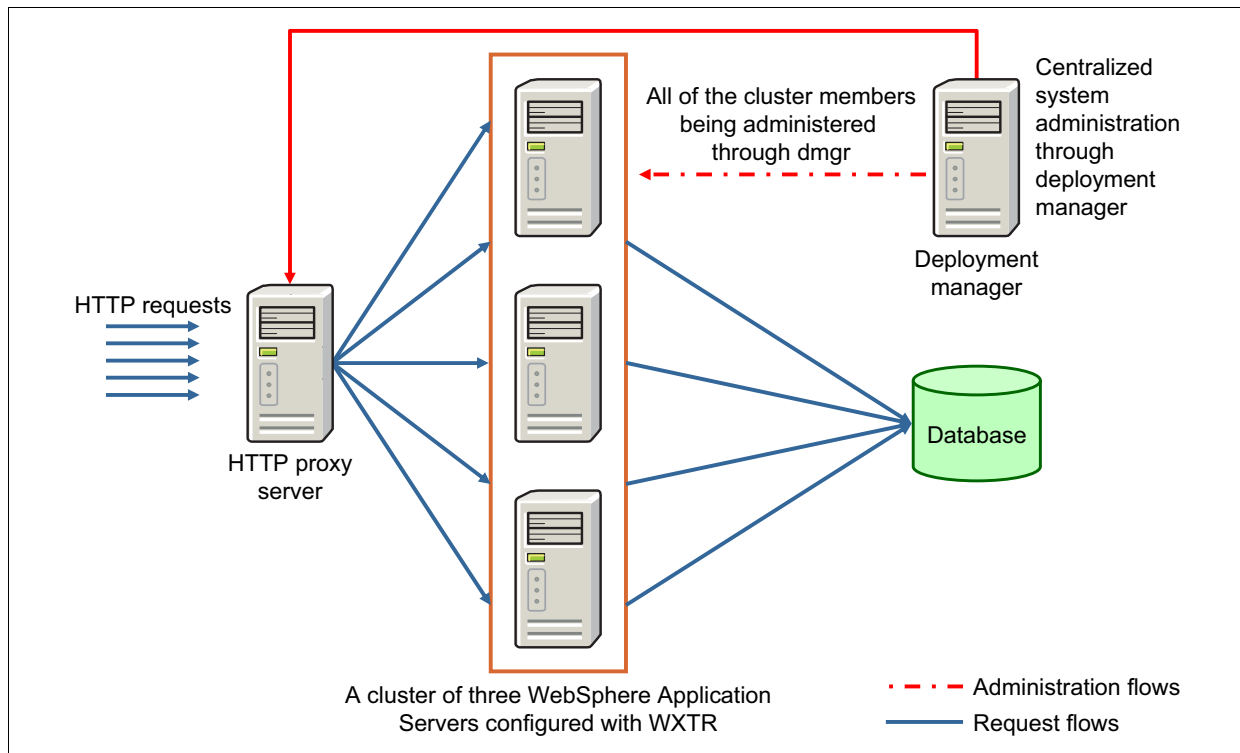


Figure 2-6 Cluster of application servers under one deployment manager

Topologies for production purposes

Production deployments are expected to handle heavy workloads and must be capable of scaling up or out when needed. They also must be highly available to ensure business continuity. To ensure this capability, you can create cluster of WebSphere Application Servers, each configured with WXTR. These clusters are alternatively referred to as *WXTR clusters*. The following types of clustered WXTR deployments are possible:

- ▶ Horizontal cluster: This type of cluster (as shown in Figure 2-7 on page 17) contains a WebSphere Application Server cluster that spans across physical servers, with each cluster member configured with WXTR.
- ▶ Vertical cluster: This type of cluster contains a WebSphere Application Server cluster that is created on a single physical server, with each cluster member configured with WXTR.

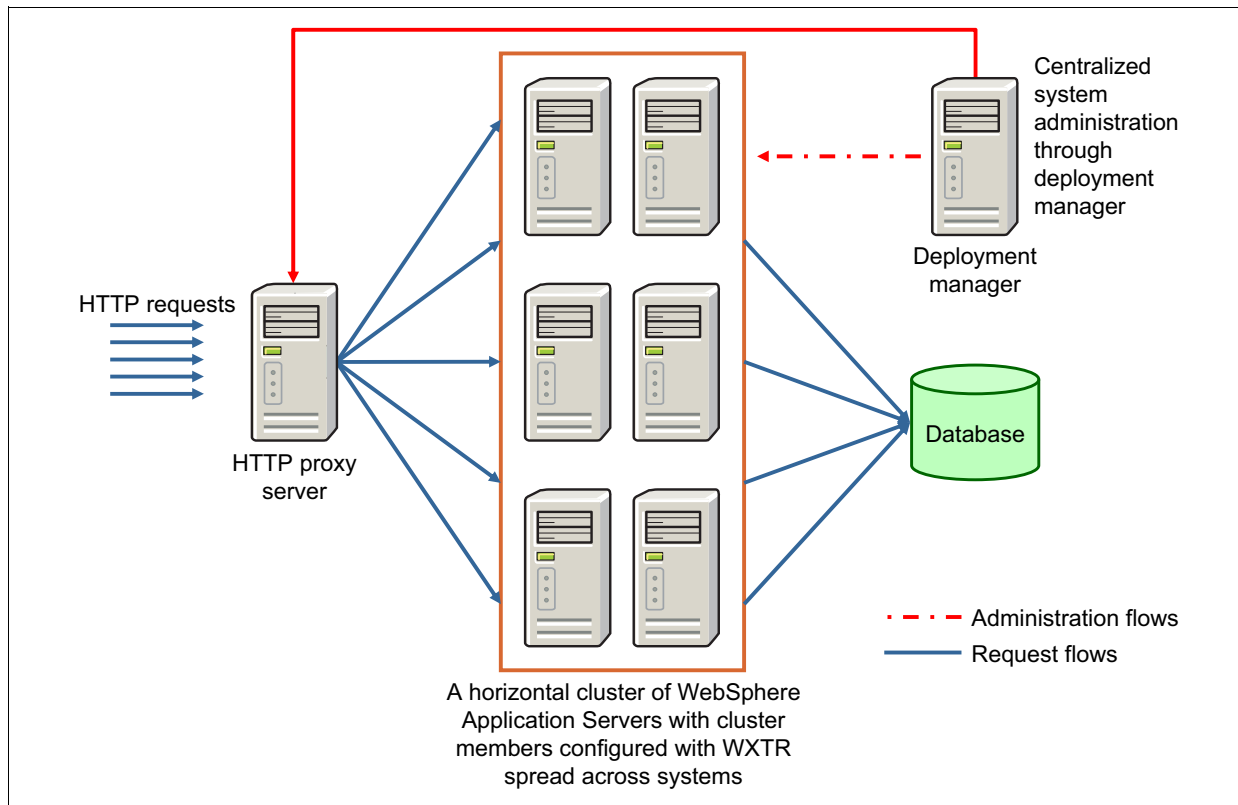


Figure 2-7 Horizontal cluster topology

Workload management in clustered environment

In any WebSphere Application Server clustered deployment, edge components, such as an HTTP Server or proxy server, act as the request-routing components. They also handle workload management by distributing the workload to individual cluster members that are based on specified criteria.

WXTR V2.1 supports the use of HTTP servers and proxy servers as edge components. By using these edge components, it is possible to perform workload management across the cluster members in a WXTR cluster.

For more information about workload management in WebSphere Application Server, see the product information center at this website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.iseries.doc/info/iseriessnd/ae/crun_srvgrp.html

High availability of a WXTR container

WXTR V2.1 introduces a heartbeat mechanism to enable high availability of a WXTR container.

When any server that is configured with WXTR is started, the heartbeat mechanism is established between WebSphere Application Server and WXTR. If the WXTR container shuts down abruptly during run time, the heartbeat mechanism in WebSphere Application Server tries to restart it. As part of this restart, the WXTR container tries to recover in-doubt transactions. After the recovery is complete, the container starts and begins to take new requests. This ability helps preserve high availability by responding to abnormal shutdown scenarios.

Important: If you are using other WebSphere products with WebSphere Application Server and WXTR, you must consider topologies to suit them as well.

2.1.4 Global transaction support

Consider a scenario in which you need to buy an airline ticket from Bangalore to Houston. You cannot get a direct flight, so you decide to break up the journey with a stop in Paris. Unless you can successfully reserve a ticket from Bangalore to Paris and another ticket from Paris to Houston, you will not make the trip. That is, you will roll back your decision to go to Houston because having a confirmed seat for only one part of the trip is not useful to you.

There are many such situations in which several smaller transactions are required to complete one primary transaction. If there is a problem with one of the smaller transactions, you would not commit the primary transaction and would want to roll things back so that none of the smaller transactions are committed, either. You want the smaller transactions managed and coordinated as a single logical unit. This type of coordinated transaction is known as a *global transaction* or *distributed transaction*.

A *global transaction* is a group of related transactions that are managed in a coordinated way. The group of transactions work as a single, logical unit of work (LUW). The transactions that constitute a global transaction can be in the same database or in different databases. Each individual transaction within a global transaction is referred to as a *transaction branch*.

The component that coordinates the smaller transactions is called the Transaction Manager. The individual data manager that participates in each transaction branch is known as the Resource Manager.

WXTR supports global transactions in the following ways:

- ▶ Lock sharing of DB2
- ▶ Two-phase commit (2PC)

Global transaction support with lock sharing of DB2

This method of global transaction support is shown in Figure 2-8 on page 19.

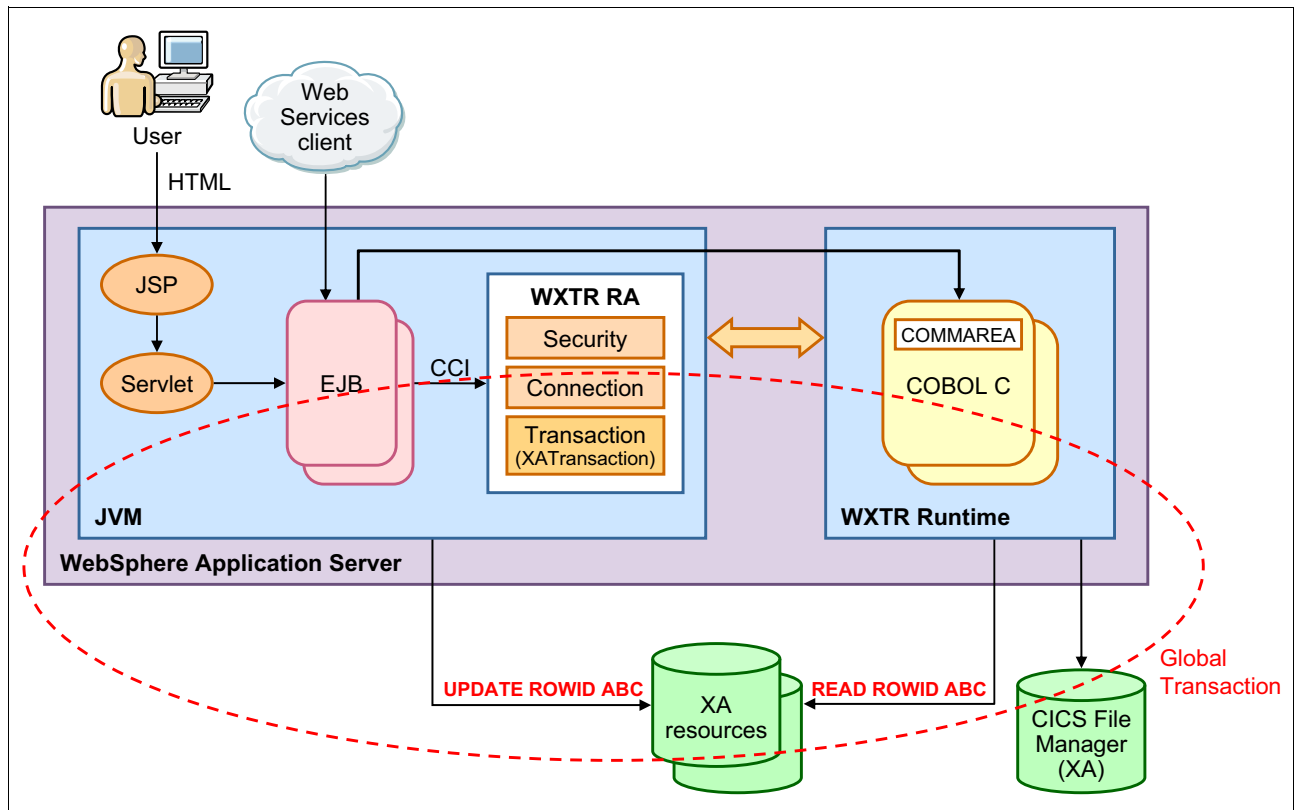


Figure 2-8 Global transaction support with lock sharing of DB2

Consider a transactional application that consists of an enterprise java bean (EJB) that is running on WebSphere Application Server and a native COBOL or C program that is running in a WXTR run time. The EJB invokes the native application through the standard interfaces that are provided by the WXTR resource adapter. Therefore, the global transaction comprises two transaction branches, one from the EJB and the other from the native application.

Let us assume that both transaction branches are trying to update the same table and row in a DB2 database. Typically, DB2 considers these requests as though they originated from two different database clients because the two transaction branches use different database connections. The EJB that makes the first update request acquires the lock for the row, and succeeds in updating the row. When the native application tries to access the same row, it is not allowed to do so because the row is already locked. So the native application waits for the lock to be released. This state results in a deadlock. If the native application performs only a read operation, it will not see the data that was updated by the EJB because that transaction branch is not yet committed. Therefore, the update is not completed.

DB2 introduced a lock sharing feature to avoid this inconsistency. The sharing feature allows database locks to be shared across multiple transaction branches. If this feature is enabled in the DB2 instance to which the WXTR run time and WebSphere Application Server are connected, global transactions can be achieved.

Important: To enable the lock-sharing feature by using Rational Application Developer Version 8.0.3 or later, open the Resources tab of the EJB deployment descriptor editor and set the Branch Coupling property in the WebSphere Extensions panel to **Tight**. The resource reference section in the deployment descriptor will look like the following example:

```
<resource-ref>
  <res-ref-name>jdbc/myDS</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
  <branch-coupling>Tight</branch-coupling>
</resource-ref>.
```

Global transaction support with two-phase commit

The two-phase commit (2PC) scenario is shown in Figure 2-9.

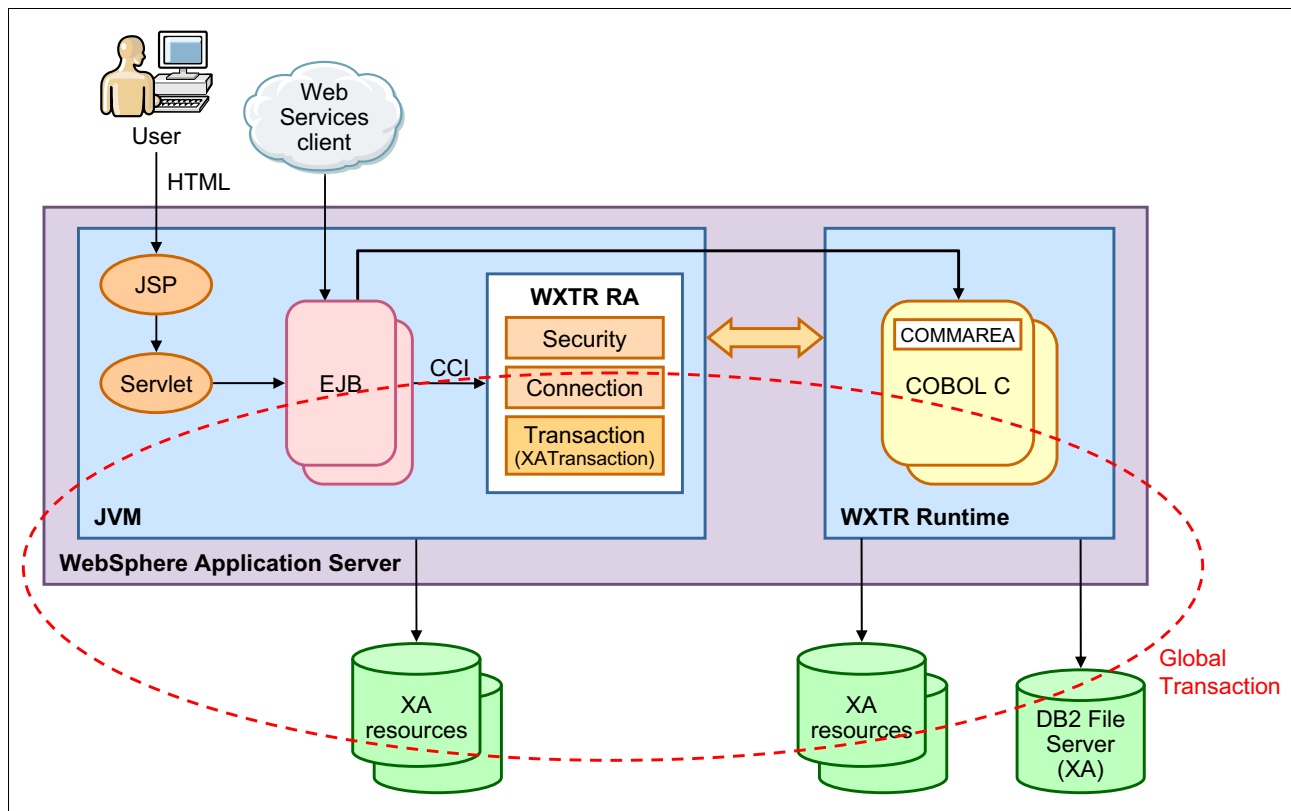


Figure 2-9 Global transaction scenario with two-phase commit

Consider the same transaction application that is described in “Global transaction support with lock sharing of DB2” on page 18. But, assume that instead of needing to share the table and row data, the two transaction branches are trying to access two logically independent, XA-compliant resources. Unlike the previous case, the resource manager is not restricted to be DB2 alone. Instead, the resource manager can be any resource that adheres to the XA specification.

In this example, we have two smaller transactions that act together as a single logical transaction. One transaction is performed by the EJB, which accesses one of the XA-compliant resources, and the other transaction is performed by the WXTR run time, which accesses the next XA resource. To coordinate these two transactions, WebSphere Application Server acts as the Transaction Manager. The WXTR resource adapter supports the 2PC protocol. WebSphere Application Server also uses the 2PC protocol to perform the global transaction.

Because WebSphere Application Server is the primary transaction coordinator, global transaction support can be achieved by using a container-managed transaction or a bean-managed transaction.

2.1.5 Identity propagation

Java EE applications that are running in WebSphere Application Server are bound by the security policies that are dictated by the application server's security infrastructure. When WXTR is brought into this environment, the Java EE applications are extended through calls to the corresponding COBOL or C applications that are hosted on WXTR. In such a heterogeneous computing environment, transactions can be initiated outside of the environment, but eventually run within WXTR.

WebSphere Application Server and WXTR maintain their own security policies and repositories. So, there must be an association between the identity of the user that starts the transaction and the user under which the transaction runs within the WXTR.

Important: For more information about the security infrastructure that is supported by WebSphere Application Server, see *WebSphere Application Server V7.0 Security Guide*, SG247660, which is available at this website:

<http://www.redbooks.ibm.com/abstracts/sg247660.html>

A high-level view of the security infrastructure that is provided by WXTR for identity propagation is shown in Figure 2-10.

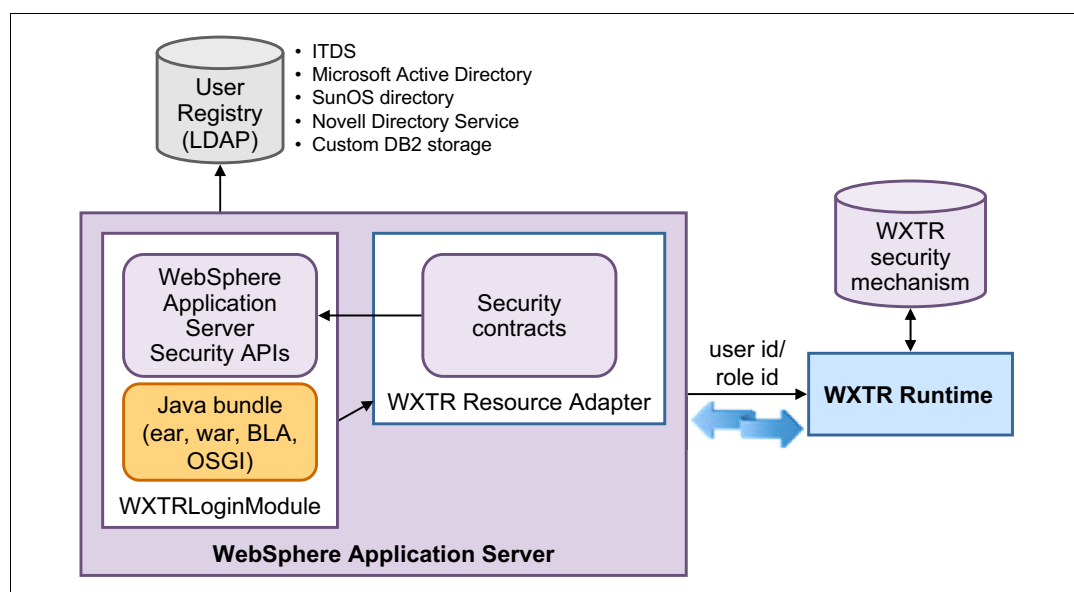


Figure 2-10 High-level view of WXTR security infrastructure

WebSphere Application Server uses the user registry to authenticate a user. The WXTR resource adapter that connects WebSphere Application Server and WXTR implements the security contracts according to the JCA specifications, which allow the propagation of credentials that are attached with the authenticated user.

The authentication of a user is carried out by WebSphere Application Server. The authorization of WXTR resources is done by WXTR. This type of user identity propagation provides end-to-end security and consistent accountability when the applications are composite with Java EE and COBOL applications.

The security in WXTR is achieved through the use of resource level keys and access to an external security manager (ESM).

Resource security level (RSL) validation provides security for access from application programs to resources during run time. WXTR uses security keys to perform security checks that determine whether you have permission to access a resource. You can specify for each resource whether WXTR will perform these checks on individual transactions by setting the RSLCheck attribute in the entry for the resource.

2.2 CICS data management support

WXTR provides support for CICS data management facilities for data that is stored in a DB2 database. These CICS data management facilities protect business applications from data loss or corruption by preserving data integrity, providing data reliability, and allowing for flexibility in the distribution of data.

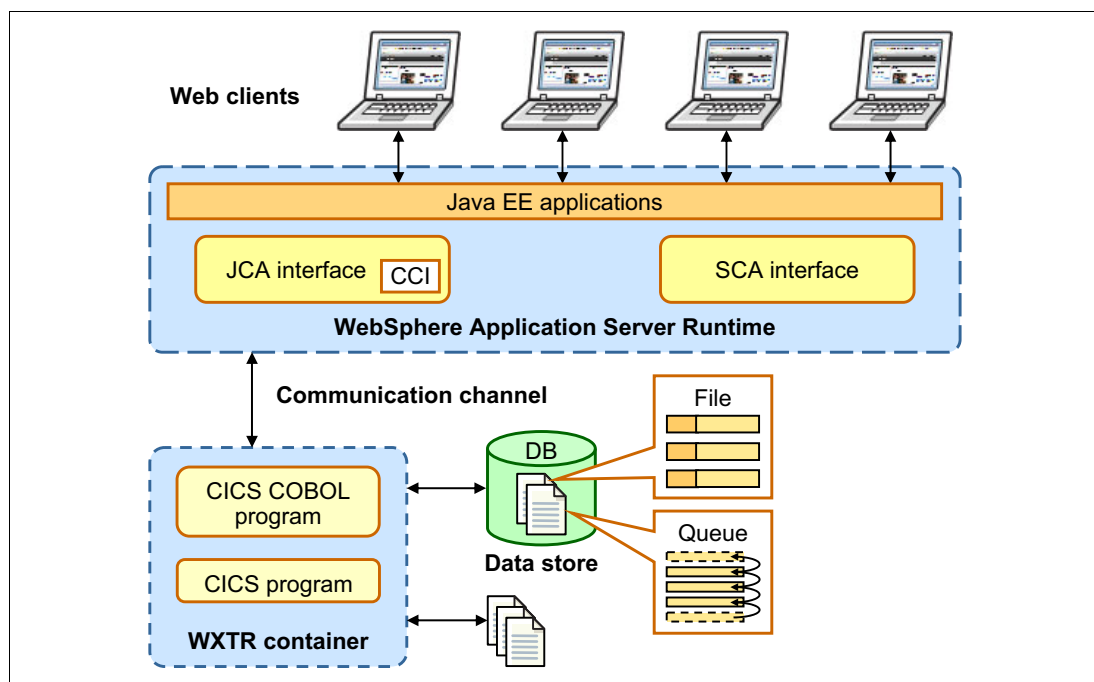


Figure 2-11 COBOL application data support

WXTR supports the Virtual Storage Access Method (VSAM) for files that are stored in DB2. WXTR also allows the user to store COBOL or C applications data in temporary storage queues (TSQ) and transient data queues (TDQ) and then use the data in subsequent transactions. The TSQ and TDQ are depicted as the Queue in Figure 2-11 on page 22.

The CICS data management facilities that are supported by WXTR are described in the next section. The configuration steps for accessing the resources also are described and sample code snippets for accessing the resources are provided.

2.2.1 VSAM file system by using DB2

The following file organizations can be used in CICS C or COBOL application programs that access VSAM-style file data:

- ▶ Key-sequenced data set (KSDS)
- ▶ Entry-sequenced data set (ESDS)
- ▶ Relative record data set (RRDS)

Key-sequenced data set

Each record in a KSDS file is identified by a key field that resides in a predefined position within the record. The records in a file are sorted automatically according to the value of their keys. This configuration groups records with the same and adjacent keys and helps reduce the cost of searching for ranges of records.

A KSDS file can be created by using the `cicsddt` tool that is shipped with WXTR, as shown in Example 2-1 on page 24. The `cicsddt` tool connects to the DB2 database and creates VSAM-style files.

Important: Before the `cicsddt` tool is run, make sure that the following environment variables are set because the tool uses these settings to connect to the DB2 database for accessing the KSDS file:

- ▶ `DB2INSTANCE=<DB2 instance name>`
- ▶ `DB2DBDFT=<DB2 database name>`
- ▶ `CICSDB2CONF_CONNECT_USER=<Username for DB2 database authentication>`
- ▶ `CICSDB2CONF_CONNECT_USING=<Password of the DB2 database user>`

Example 2-1 Creating a KSDS-type VSAM file in DB2 by using the cicsddt tool

```
#cicsddt -c create KSDSFILE

CWXTD0068I/0500: CICSDDT talking to: 'wxtrdb'
CWXTD0069I/0501: Version [1.0 : 25-07-1995]
CWXTD0070I/0502: Contacting Database...
CWXTD0172I/0738: (Database OK)
cicsddt: -> create

[Table Name.....: KSDSFILE

[File Type [Ksds/Esds/Rrds] .: K[sds]
                                (Taking default: KSDS)
[Column 01: Name .....: UID
[Column 01: Type .....: char
[Column 01: Size .....: 8
[Column 02: Name .....: NAME
[Column 02: Type .....: char
[Column 02: Size .....: 20
[Column 03: Name .....:
[Primary Index Name .....: CICS.KSDSFILE0
[Index Part 01: Column Name .: UID
[Index Part 01: Ordering ....: a[scending]
[Index Part 02: Column Name .:
cicsddt: -> quit
```

Because the CICS style COBOL or C applications use logical names to access KSDS files, you must use the wsadmin scripting tool to add a file definition (FD) entry for each file to the WXTR container, as shown in Example 2-2. The wsadmin tool that is included with WebSphere Application Server is extended in WXTR to provide this facility. For more information about the wsadmin tool in WXTR, see 2.3.2, “Using the command line” on page 32.

Example 2-2 Adding FD entry for KSDS file to WXTR container by using wsadmin tool

```
#!/wsadmin.sh -lang jython
WASX7209I: Connected to process "server1" on node apnaNode01 using SOAP connector;
The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>AdminTask.cics('[-command add -nodeName apnaNode01 -serverName server1 -c
fd -name KFILE -properties [[BaseName KSDSFILE] [IndexName KSDSFILE0] [OpenStatus
open] [ActivateOnStartup yes]]')
,,
wsadmin>AdminConfig.save()
```

KFILE is the FD entry name and KSDSFILE is the actual file name, as shown in Example 2-1 on page 24. For adding the FD entry, the wsadmin tool also requires the nodeName and serverName of the node and server where WXTR is installed.

Restart the WXTR container to reflect the changes that were made to the FD entries.

CICS application developers can use the FD entry (KFILE) to access the records from the file, as shown in Example 2-3 on page 25.

Example 2-3 Accessing the KSDS file records from CICS COBOL program

```
EXEC CICS READ FILE('KFILE')
                                INTO(USER-REC)
                                RIDFLD(USER-ID)
                                LENGTH(LENGTH OF USER-REC)
                                RESP(USER-RESP)

END-EXEC
```

WXTR also supports offline access to these KSDS files from non CICS COBOL applications, as shown in Example 2-4.

Important: The following environment variables must be set in the environment where the non CICS programs are executed:

- ▶ COBRTOPT=FILESYS=DB2
- ▶ COBOL_DB2_DBNAME=<Database Name>
- ▶ COBOL_DB2_USERNAME=<Database Username>
- ▶ COBOL_DB2_USERPASS=<Database Password>
- ▶ COBOL_DB2_DBPREFIX=cics

Example 2-4 Accessing KSDS files from non CICS COBOL programs offline

```
FILE-CONTROL.
    SELECT RECORD-FILE
        ASSIGN TO 'CICS.KFILE'
        ORGANIZATION IS INDEXED
        ACCESS MODE IS SEQUENTIAL
        RECORD KEY IS KSDSFILE0
        STATUS IS RECORD-STATUS.

PROCEDURE DIVISION.
    READ RECORD-FILE.
```

Entry-sequenced data set

Records in an ESDS file are stored in the sequence in which they are written to the file. New records are appended to the end of the file. When records are deleted, the disk space that they used is not automatically reclaimed or reused. You can reclaim the disk space by reorganizing the file.

Entry-sequenced files often are used when the records are to be accessed in the order in which they were written, such as with log files and audit trail files.

Like KSDS files, ESDS files can be created by using the `cicsddt` tool that is shipped with WXTR (see Example 2-5 on page 26).

Example 2-5 Creating an ESDS-type VSAM file in DB2 by using the cicsddt tool

```
# cicsddt -c create ESDSFILE

CWXTD0068I/0500: CICSDDT talking to: 'wxtrdb'
CWXTD0069I/0501: Version [1.0 : 25-07-1995]
CWXTD0070I/0502: Contacting Database ...
CWXTD0172I/0738: (Database OK)
cicsddt: -> create
[Table Name.....: ESDSFILE
[File Type [Ksds/Esds/Rrds] .: E[sds]
[Column 01: Name .....: RBA
[Column 01: Type .....: char
[Column 01: Size .....: 4
[Column 02: Name .....: LOGINFO
[Column 02: Type .....: char
[Column 02: Size .....: 100
[Column 03: Name .....:
cicsddt: -> quit
```

Next, use the wsadmin tool to add an FD entry for each file to the WXTR container, as shown in Example 2-6. This addition allows access to the ESDS files from CICS COBOL or C applications.

Example 2-6 Adding an FD entry for an ESDS file to the WXTR container by using wsadmin tool

```
# ./wsadmin.sh -lang jython
WASX7209I: Connected to process "server1" on node apnaNode01 using SOAP connector;
The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>AdminTask.cics('[-command add -nodeName apnaNode01 -serverName server1 -c
fd -name EFILE -properties [[BaseName ESDSFILE] [IndexName ESDSFILE0] [OpenStatus
open] [ActivateOnStartup yes]]')
,,
wsadmin>AdminConfig.save()
```

In Example 2-6, EFILE is the FD entry name and ESDSFILE is the actual file name, as shown in Example 2-5. To add the FD entry, wsadmin tool also requires the nodeName and serverName on which WXTR is installed.

When you are finished creating the FD entries, restart the WXTR container to reflect the changes.

CICS COBOL application developers can use the FD entry, EFILE, in the CICS file APIs in application programs to access the records from the file as shown in Example 2-7.

Example 2-7 Accessing ESDS file records from a CICS COBOL application

```
EXEC CICS WRITE FILE('EFILE')
                                FROM(LOG-DETAILS)
                                RIDFLD(ESDSFILE0)
                                RBA
END-EXEC.
```

Example 2-8 shows how non CICS COBOL applications read records from ESDS files in offline mode.

Example 2-8 Accessing an ESDS file from a non CICS COBOL application offline

```
FILE-CONTROL.
    SELECT RECORD-FILE
        ASSIGN TO 'CICS.ESDSFILE'
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUENTIAL
        STATUS IS RECORD-STATUS.
PROCEDURE DIVISION.
    READ RECORD-FILE.
```

Relative record data set

An RRDS file is an array of fixed-length slots in which records can be stored. A record can be added to the first free slot in the file, at the end of the file, or in a specified slot in the file. Records can be fixed or have variable lengths up to the slot size that was predefined to the File Manager. You can update or delete any record. Any slot that is freed when a record is deleted can be reused by inserting another record in the free slot. The primary index is a physical part of the data in the record.

Example 2-9 shows the creation of RRDS files by using the `cicsddt` tool.

Example 2-9 Creating an RRDS-type VSAM file in DB2 by using the `cicsddt` tool

```
# cicsddt -c create RRDSFILE
CWXTD0068I/0500: CICSDDT talking to: 'wxtrdb'
CWXTD0069I/0501: Version [1.0 : 25-07-1995]
CWXTD0070I/0502: Contacting Database ...
CWXTD0172I/0738: (Database OK)
cicsddt: -> create
[Table Name.....: RRDSFILE
[File Type [Ksds/Esds/Rrds] .: R[sds]
[Column 01: Name .....: RRN
[Column 01: Type .....: char
[Column 01: Size .....: 4
[Column 02: Name .....: EMPID
[Column 02: Type .....: char
[Column 02: Size .....: 6
[Column 03: Name .....: SALARY
[Column 03: Type .....: char
[Column 03: Size .....: 10
[Column 04: Name .....:
cicsddt: -> quit
```

Next, use the `wsadmin` tool to create an FD entry for each RRDS file in the WXTR container, as shown in Example 2-10 on page 28. This creation enables access to the RRDS files from CICS COBOL or C applications.

Example 2-10 Adding an FD entry for an RRDS file to the WXTR container by using the wsadmin tool

```
# ./wsadmin.sh -lang jython
WASX7209I: Connected to process "server1" on node apnaNode01 using SOAP connecto
r; The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>AdminTask.cics('[-command add -nodeName apnaNode01 -serverName server1 -c
fd -name RFILE -properties [[BaseName RRDSFILE] [IndexName RRDSFILE0] [OpenStatus
open] [ActivateOnStartup yes]]')
''
wsadmin>AdminConfig.save()
```

In this example, RFILE is the FD entry name and RRDSFILE is the actual file name, as shown in Example 2-9 on page 27. To add the FD entry, wsadmin tool also requires the nodeName and serverName on which WXTR is installed.

When you are finished creating the FD entries, restart the WXTR container to reflect the changes.

Example 2-11 shows how CICS COBOL application developers can use the FD entry, RFILE, in the CICS file APIs in application programs to access the records from the file.

Example 2-11 Accessing an RRDS file from a CICS COBOL application

```
EXEC CICS WRITE FILE('RFILE') RRN
                                FROM(USER-REC)
                                LENGTH(LENGTH OF USER-REC)
                                RIDFLD(RRDSFILE0)

END-EXEC.
```

Example 2-12 shows how non CICS COBOL applications read records from RRDS files in offline mode.

Example 2-12 Accessing RRDS records from non CICS COBOL applications in offline mode

```
FILE-CONTROL.
    SELECT RECORD-FILE
        ASSIGN TO 'CICS.RRDSFILE'
        ORGANIZATION IS RELATIVE
        ACCESS MODE IS DYNAMIC
        RELATIVE KEY IS RELATIVE-KEY
        LOCK IS AUTOMATIC
        STATUS IS RECORD-STATUS.
PROCEDURE DIVISION.
    READ RECORD-FILE.
```

2.2.2 Temporary storage queues

Temporary storage queues (TSQs) are used for shared reading, writing, and updating of data by multiple transactions. Transactions can write, update, read, and delete data in a temporary storage queue multiple times until the queue is deleted.

Temporary storage queues are not predefined to a WXTR container. They are created the first time that you write to a queue by using a symbolic name. Any transaction can retrieve temporary data by using the symbolic name that is assigned to the queue. Specific records within a queue are referred to by their relative position numbers. TSQs provided by WXTR are recoverable.

Example 2-13 shows how CICS COBOL applications can use TSQs for storing temporary data. The TSQ named EMPQUEUE is created when the queue is accessed at first time.

Example 2-13 Using TSQs in a CICS COBOL application program

```
EXEC CICS WRITEQ TS
                                QUEUE      ('EMPQUEUE')
                                FROM        (MSG)
                                LENGTH      (80)
                                RESP        (EMP-RESP)
END-EXEC.
```

2.2.3 Transient data queues

Transient data queues (TDQs) provide a generalized queuing facility in which programmers can store application data for subsequent internal or external processing. The following types of TDQs are supported:

- ▶ Intrapartition
- ▶ Extrapartition

Application programs use *intrapartition* TDQs to queue data that is to be processed by other programs that are running as separate tasks within the same WXTR container. Data that is directed to or from an external destination is called *extrapartition data* and consists of sequential fixed-length or variable-length records. The record format for an extrapartition destination must be defined in the TDQ. Extrapartition transient data files are not recoverable. In general, extrapartition TDQ data is intended for subsequent input to non CICS programs.

Example 2-14 and Example 2-15 on page 30 show how to add the TDQ entry to the WXTR container by using the wsadmin tool. If properties are not specified to the TDQ, an intrapartition queue is created by using the specified TDQ name by default.

Example 2-14 Adding an intrapartition TDQ entry to the WXTR container

```
# ./wsadmin.sh -lang jython
WASX7209I: Connected to process "server1" on node apnaNode01 using SOAP connector;
The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"

wsadmin>AdminTask.cics('[-command add -nodeName apnaNode01 -serverName server1 -c
tdd -name QUE1]')
,,

wsadmin>AdminConfig.save()
```

Example 2-15 Adding an extrapartition TDQ entry to the WXTR container

```
# ./wsadmin.sh -lang jython
WASX7209I: Connected to process "server1" on node apnaNode01 using SOAP connector;
The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>AdminTask.cics('[-command add -nodeName apnaNode01 -serverName server1 -c
tdd -name TTDQ -properties [[DestType extrapartition] [ExtrapartitionFile
TTDQ.txt] [RSLKey public] [RecordLen 1024]]]')
''
wsadmin>AdminConfig.save()
```

When you are finished, restart the WXTR container to reflect the changes.

Example 2-16 shows how to access TDQ records from CICS COBOL applications.

Example 2-16 Accessing TDQ records from a CICS COBOL application

```
EXEC CICS WRITEQ TD
                                QUEUE    (TTDQ)
                                FROM      (EMP-MSG)
                                LENGTH    (1024)
                                RESP      (EMP-RESP)

END-EXEC.
```

2.3 Unified administration

This section focuses on the administration methods that are available for managing Java and COBOL assets in WXTR.

WXTR provides a common facility for managing COBOL and Java EE assets through the standard WebSphere Application Server administration framework. The administrative console can be used to deploy Java EE applications (those applications that are developed by using the WXTR-supported JCA or SCA standard interfaces) on WebSphere Application Server. The console also can be used to set the WXTR container parameters to run COBOL applications.

2.3.1 Using the administrative console

This section describes how to administer Java EE and COBOL or C assets in WXTR by using the WebSphere Application Server administrative console.

The administrative console is a graphical interface that allows you to manage applications and perform system administration tasks. It runs in a web browser. Figure 2-12 on page 31 shows the administrative console's application configuration window. To view this page, perform the following steps:

1. Start the WebSphere Application Server administration console by entering the following URL: `http://hostname:port/admin` where `hostname` is the name or IP address of the machine where WebSphere Application Server is running and `port` is the WebSphere Application Server administration port.

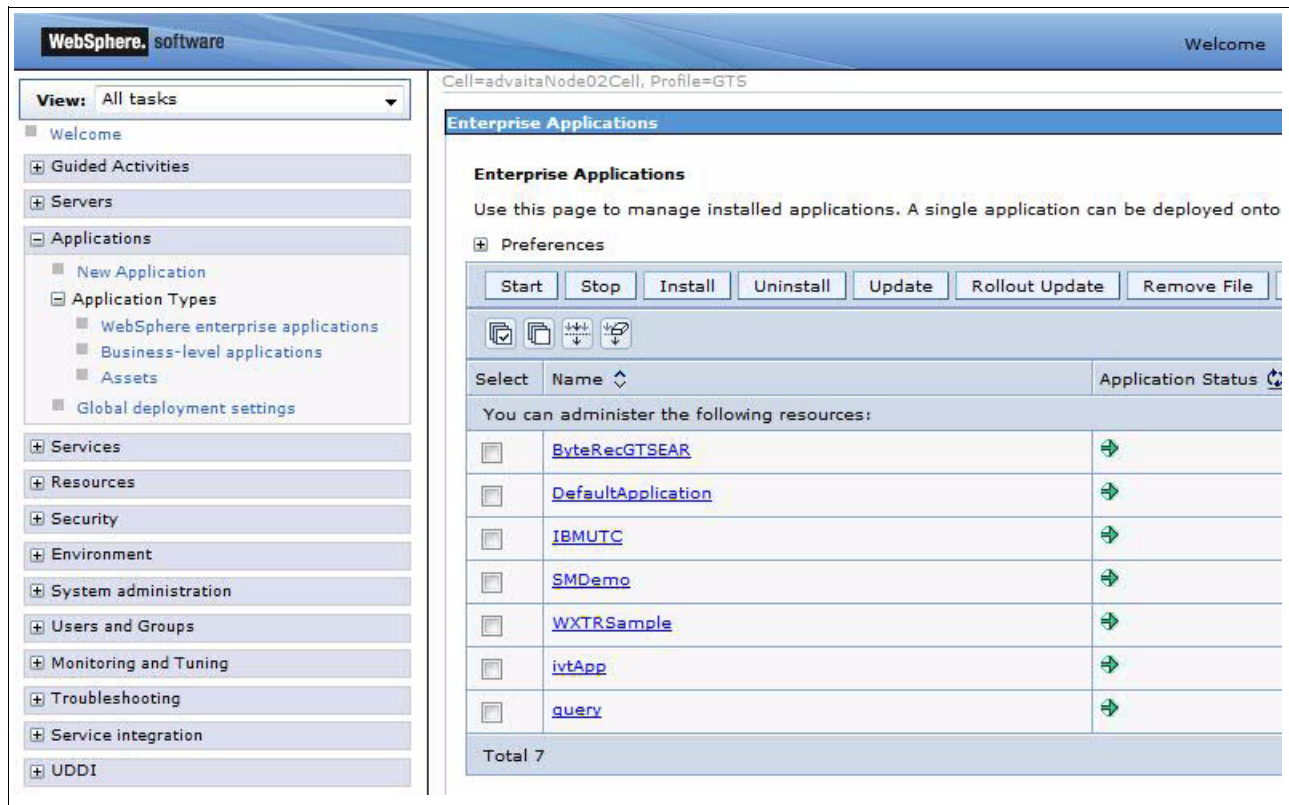


Figure 2-12 Application configuration window of the WebSphere Application Server administrative console

2. Select **Applications** → **Application Types** → **WebSphere enterprise applications**.

Managing Java EE assets

Using the administrative console, administrators can perform the following operations with Java EE assets:

- ▶ Deploy new applications to a server
- ▶ Start and stop applications and modify certain configurations
- ▶ Add and delete Java EE resources for applications that require data access
- ▶ Obtain product version information (on the opening window of the console)

For more information about the administrative capabilities that are provided by the administrative console, see the WebSphere Application Server Information Center at this website:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp>

Managing WXTR container assets

The administrative console can also be used to configure properties for the WXTR container. To view the console's WXTR container settings page and set these properties, complete the following steps:

1. Select **Servers** → **Server Types** → **WebSphere Application Servers**.
2. Choose the name of the server of which WXTR container settings you want to view.

3. Select **Configuration** → **WebSphere eXtended Transaction Runtime Container Settings** → **WXTR container** to configure properties such as the maximum and minimum number of processes that WXTR uses to run transactions, the port number to be used for internal communication, and the database authentication information.
4. Select **Configuration** → **WebSphere eXtended Transaction Runtime Container Settings** → **WXTR pool settings** to configure the properties of the different memory pools that are used by WXTR container.
5. Select **Configuration** → **WebSphere eXtended Transaction Runtime Container Settings** → **WXTR file manager settings** to configure the properties of the files that are used by WXTR container.
6. Select the wanted container setting and change its value as needed.
7. When finished, restart the WXTR container to reflect the modified settings values.

Figure 2-13 shows the WXTR container settings page of the administrative console.

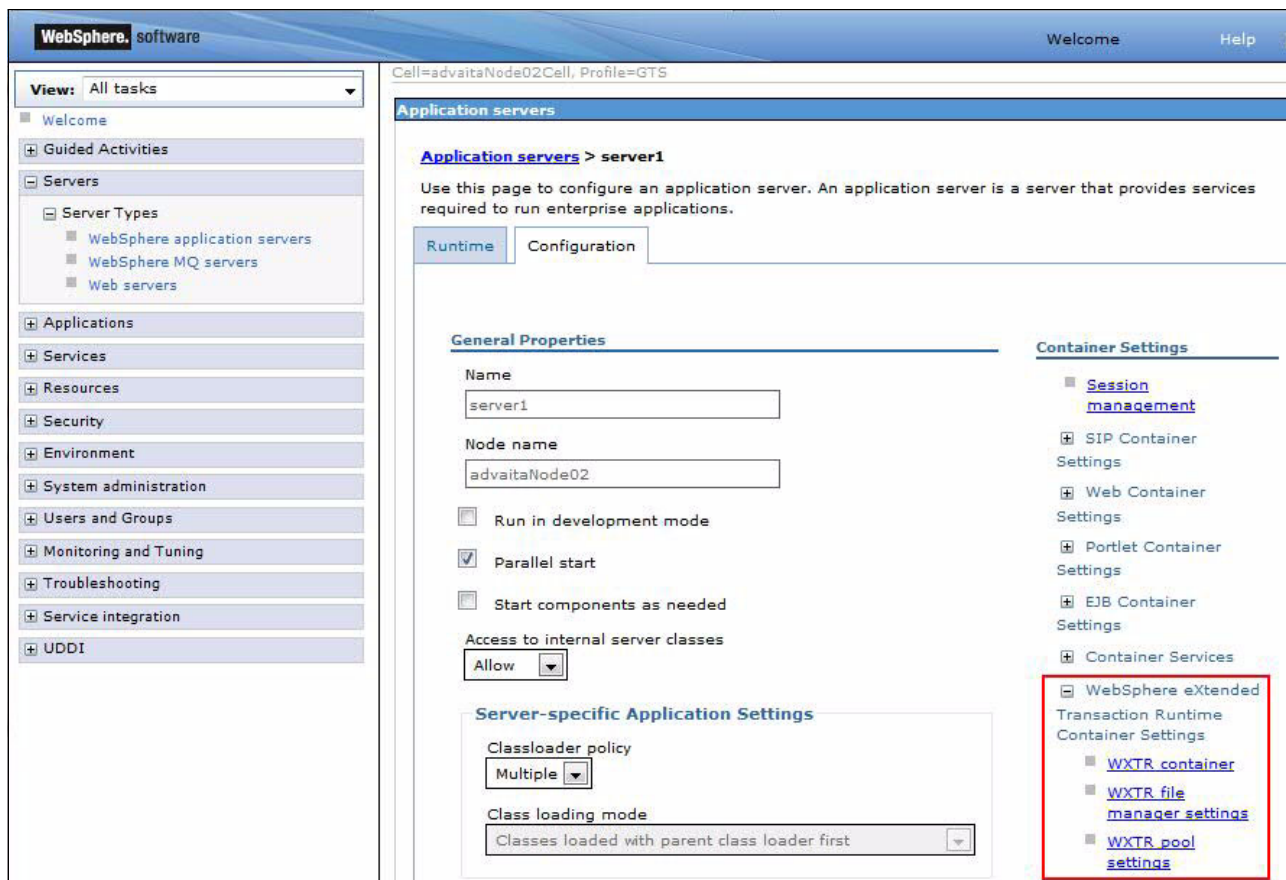


Figure 2-13 WXTR container setting page in WebSphere Application Server administration console

2.3.2 Using the command line

WebSphere Application Server can be configured and managed from the command line by using the wsadmin scripting tool.

The wsadmin tool is a non-graphical alternative to the administrative console for configuring and managing WebSphere Application Server. The tool supports the Jacl and Jython scripting languages. WXTR supports only the Jython scripting language.

The following wsadmin objects are available to use with scripts:

- ▶ AdminControl: Use to run operational commands
- ▶ AdminConfig: Use to run commands to create or modify WebSphere Application Server configuration elements
- ▶ AdminApp: Use to administer applications
- ▶ AdminTask: Use to run administrative commands
- ▶ Help: Use to obtain general help

Scripts use these wsadmin objects to communicate with MBeans that run in WebSphere Application Server processes. MBeans are Java objects that represent Java Management Extensions (JMX) resources.

Important: If security is enabled for a particular application server, you must supply authentication information to the wsadmin tool so it can communicate with the server.

WXTR supplies the **cics** command with the AdminTask object of the wsadmin scripting tool. Use the command to add, update, get, or delete WXTR container configuration properties, as shown in Example 2-17.

Example 2-17 Using the WXTR-supplied cics command to generate AdminTask help output

```
# ./wsadmin.sh -lang jython
WASX7209I: Connected to process "server1" on node apnaNode01 using SOAP connector;
The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>print AdminTask.help('cics')
WASX8006I: Detailed help for command: cics
Description: Command for all WXTR configuration commands
Target object: None
Arguments:
  *command - Operation to be performed. Eg: add, delete, update etc
  *c - CICS resource class on which command is to be executed.
  colonSeperatedOutput - Display the output as colon separated list.
  colonSeperatedEntireListAsOutput - Fetches all the entries under the specified
resource class and displays them as a colon separated list
  name - Name of the CICS resource.
  *nodeName - Name of the node where WebSphere Application Server configured with
WXTR resides.
  *serverName - Name of WebSphere Application Server configured with WXTR.
  properties - CICS command properties
Steps:
  None
```

Example 2-18 on page 34 shows how to use the **cics** command from within the wsadmin scripting tool to add an intrapartition TDQ entry to a WXTR container.

Example 2-18 Adding an intrapartition TDQ entry to a WXTR container using the cics admin command

```
# ./wsadmin.sh -lang jython
WASX7209I: Connected to process "server1" on node apnaNode01 using SOAP connector;
The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>AdminTask.cics('[-command add -nodeName apnaNode01 -serverName server1 -c
tdd -name ULOG]')
''
wsadmin>AdminConfig.save()
```

When you are finished, restart the WXTR container to reflect the updates.

The TDQ entry that was added in Example 2-18 can be viewed with the wsadmin scripting tool by using the **get** option of the **cics admin** command, as shown in Example 2-19.

Example 2-19 Viewing TDQ entry of WXTR container with get command of cics admin command

```
# ./wsadmin.sh -lang jython
WASX7209I: Connected to process "server1" on node apnaNode01 using SOAP connector;
The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>AdminTask.cics('[-command get -nodeName apnaNode01 -serverName server1 -c
tdd -name ULOG]')
'\nULOG:\nGroupName=""\nActivateOnStartup=yes\nResourceDescription="Transient Data
Definition"\nAmendCounter=0\nPermanent=no\nRemoteSysId=""\nRemoteName=""\nRSLKey=p
rivate\nDestType=intrapartition\nIOMode=output\nExtrapartitionFile=""\nWhenOpened=
at_startup\nOpenMode=truncate\nRecordType=fixed_length\nRecordLen=1024\nRecordTerm
inator=0\nIndirectQueueId=""\nFacilityType=file\nRecoveryType=logical\nTriggeredTr
ansId=""\nTriggerLevel=0\nFacilityId=""\nMaxSize=0\nTemplateDefined=no\n'
```

The TDQ type can be changed from intrapartition to extrapartition by using the **update** command of the **cics admin** command, as shown in Example 2-20.

Example 2-20 Changing the TDQ type from intrapartition to extrapartition using the update command

```
# ./wsadmin.sh -lang jython
WASX7209I: Connected to process "server1" on node apnaNode01 using SOAP connector;
The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>AdminTask.cics('[-command update -nodeName apnaNode01 -serverName server1
-c tdd -name ULOG -properties [[DestType extrapartition] [ExtrapartitionFile
ULOG.txt] [RSLKey public] [RecordLen 1024]]]')
''
wsadmin>AdminConfig.save()
```

The TDQ can be deleted with the **delete** command, as shown in Example 2-21 on page 35.

Example 2-21 Deleting the TDQ by using the delete command

```
# ./wsadmin.sh -lang jython
WASX7209I: Connected to process "server1" on node apnaNode01 using SOAP connector;
The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>AdminTask.cics('[-command delete -nodeName apnaNode01 -serverName server1
-c tdd -name UL0G]')
''
wsadmin>AdminConfig.save()
```

2.3.3 Using a Jython script

The wsadmin tool can accept a Jython script file as an input argument.

Commands to add, update, delete, or get WXTR container resource definitions can be written in a Jython script file that is passed as input to the wsadmin tool. The use of a script file reduces administration efforts when multiple resource definitions need to be added to a container. If any of the commands fail, the tool raises an exception with an error message. These exceptions need to be handled properly in scripts by using try and except blocks.

Example 2-22 shows how to delete a file definition (FD) entry from a WXTR container and then add the same FD entry. Because AdminTask.cics raises an exception if the entry does not exist in the WXTR container, the delete task is included in a try block. If an exception arises, it is handled and the corresponding error is raised in the except block. If the entry exists, it is deleted and then added with the same name.

Example 2-22 Sample Jython script to delete and then add an FD entry to a WXTR container

```
import sys
node = server = FDentry = basename = indexname = ''
node = sys.argv[0]
server = sys.argv[1]
FDentry = sys.argv[2]
basename = sys.argv[3]
indexname = sys.argv[4]
try:
    print ""
    print "Deleting the FD entry from WXTR container."
    print ""
    AdminTask.cics('[-command delete -nodeName ' + node + ' -serverName ' +
server + ' -c fd -name ' + FDentry + ' ]')
    print "Deleted the FD entry successfully from WXTR container."
    print ""
    AdminConfig.save()
except:
    print 'Error: ', sys.exc_info()[0], sys.exc_info()[1]
print "Adding the FD entry to WXTR container."
AdminTask.cics('[-command add -nodeName ' + node + ' -serverName ' + server + ' -c
fd -name ' + FDentry + ' -properties [[BaseName ' + basename + ' ] [IndexName ' +
indexna
me + ' ] [OpenStatus open] [ActivateOnStartup yes]]]')
AdminConfig.save()
print ""
print "Added the FD entry to WXTR container successfully."
```

The code that is shown in Example 2-22 on page 35 accepts the following arguments:

- ▶ Node name
- ▶ Server name
- ▶ FD entry name
- ▶ Base name
- ▶ Index name

This code can be reused to add multiple FD definitions to a WXTR container by passing the respective definition names as input arguments. The code in Example 2-22 on page 35 is saved as `/tmp/FDadd.py` and run. The results of this approach are shown in Example 2-23.

Example 2-23 Sample results from adding an FD entry named USERFILE to a WXTR container

```
# ./wsadmin.sh -lang jython -f /tmp/FDadd.py anchor3Node01 server1 USERFILE
USERINFO USERINFO0
WASX7209I: Connected to process "server1" on node anchor3Node01 using SOAP
connector; The type of process is: UnManagedProcess
WASX7303I: The following options are passed to the scripting environment and are
available as arguments that are stored in the argv variable: "[anchor3Node01,
server1, USERFILE, USERINFO, USERINFO0]"
```

Deleting the FD entry from WXTR container.

Deleted the FD entry successfully from WXTR container.

Adding the FD entry to WXTR container.

Added the FD entry to WXTR container successfully.

WXTR ships with a sample Jython script, `sample.py`, in the `/samples` directory. Refer to this sample script for developing the Jython scripts that are needed for WXTR container administration tasks that use the `wsadmin` tool.

2.4 Modernizing applications

When used with tools such as Rational Application Developer and Rational Developer for Power Systems Software™, WXTR provides a modern Java EE and COBOL or C application development experience. It allows you to develop, deploy, and debug applications within a single integrated development environment (IDE).

COBOL and C applications can be modernized and extended by using the integrated development and deployment tools of WXTR. This approach provides the following advantages:

- ▶ Developers can switch between Java, COBOL, and C perspectives in the same IDE.
- ▶ Java applications can be developed on any Windows operating system platform where Rational Application Developer is installed.
- ▶ Java applications can use WXTR-supplied code snippets to create connections to and call COBOL and C programs.
- ▶ Java applications can be deployed on WebSphere Application Server instances where WXTR is installed. WXTR supports only the AIX operating system.
- ▶ COBOL applications can be developed by using the Rational Developer for Power Systems Software in the same IDE.

- ▶ COBOL applications can be developed by using CICS COBOL templates.
- ▶ COBOL applications can be deployed in a WXTR container on a remote AIX system.
- ▶ Problems can be debugged easily because the Java and COBOL or C applications exist in the same IDE.
- ▶ Users can step into COBOL or C programs while debugging application calls to them in Java programs.

2.4.1 Using Rational Developer for Power Systems Software

IBM Rational Developer for Power Systems Software (RDp) provides an integrated set of application development tools. It extends the base functionality that is available in Eclipse, in particular Remote System Explorer (RSE).

RDp works seamlessly with other Eclipse-based tools, such as IBM Rational Team Concert™ and Rational Application Developer for WebSphere Software IDE. It allows developers to edit source code with the Remote Systems LPEX Editor, which enables automatic indenting for language parsing and text effects to emphasize different parts of the code.

The RDp source code can be stored in Eclipse projects, which offers the option of working while connected to an IBM AIX server and then synchronizing your source code when you reconnect to the AIX server. The projects also enable team work sharing with any source control provider that works with Eclipse. Any error feedback is integrated with the standard Eclipse mechanisms to quickly bring the developer to the problematic line of code.

Remote System Explorer can be used to see files and processes on an IBM AIX system. Shells can be launched in integrated windows to interact with the AIX server.

This section describes how to develop new COBOL applications with RDp and describes the templates and code snippets that are provided with WXTR to make the development process easier. In addition, it covers deployment and testing of the COBOL applications by using the sample portal that is provided by WXTR. COBOL application debugging with RDp is also explained, as is the procedure to import existing COBOL applications for enhancement. C applications can be developed similarly.

WXTR-supplied templates

WXTR supplies COBOL program templates that application developers can use to create new COBOL applications. The templates can be found in *WXTR Install Directory/samples/RDP/wxtr_snippets.xml* and must be imported into RDp before the process of creating applications begins.

Complete the following steps to import the WXTR-supplied COBOL program templates into RDp:

1. Copy the *WXTR Install Directory/samples/RDP/wxtr_templates.xml* file from the AIX machine where WXTR is installed to a Windows or Linux workstation where RDp is installed.
2. Launch RDp.
3. IBM Rational Application Developer and RDp are installed in the same IDE, so multiple perspectives are possible. Complete the following steps to open the AIX COBOL perspective:
 - a. Select **Window** → **Open Perspective** → **Other**, as shown in Figure 2-14 on page 38.

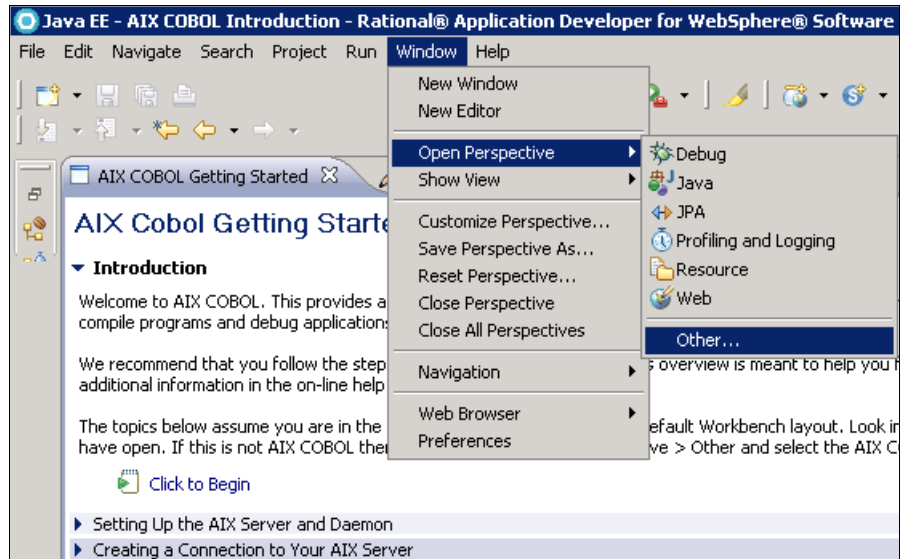


Figure 2-14 Opening the perspectives that are supported by Rational tools

- b. From the list, choose **AIX COBOL** and then click **OK**, as shown in Figure 2-15.

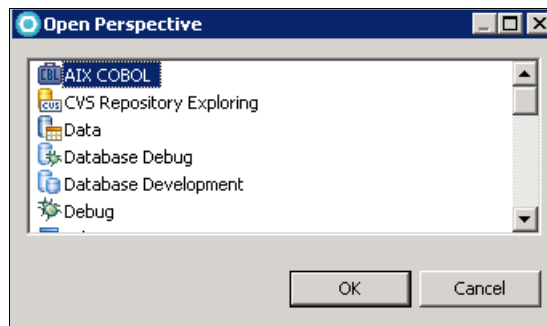


Figure 2-15 Selecting the AIX COBOL perspective

4. After designating the AIX COBOL perspective, proceed with the process of importing the COBOL program templates by selecting **Window** → **Preferences**.
5. Select **COBOL** → **AIX** → **File Templates**, as shown in Figure 2-16 on page 39.

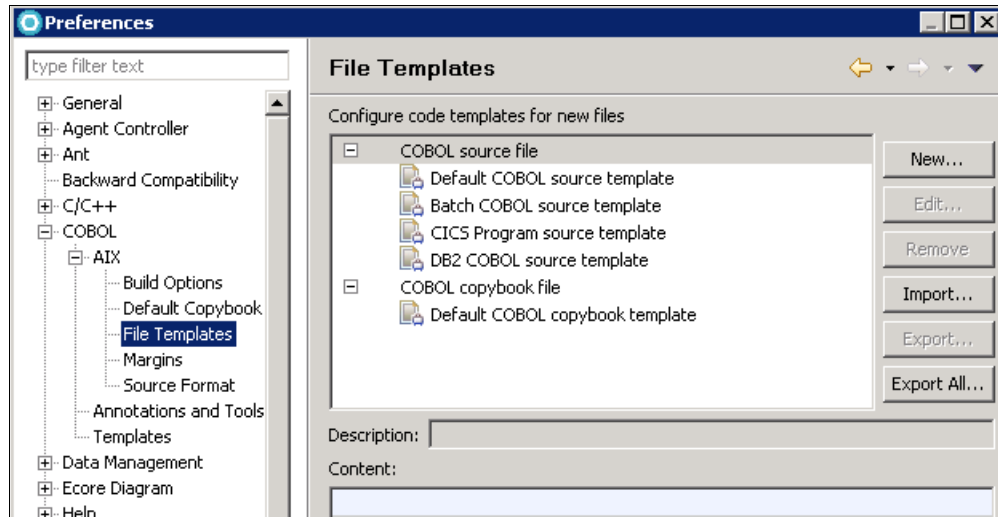


Figure 2-16 Open File Templates to import the WXTR supplied templates

6. In the File Templates pane, select **Import** to import the WXTR-supplied COBOL templates to the IDE, as shown in Figure 2-16. In the file selection window, navigate to the appropriate directory, select the wxtr_templates.xml file, and click **Open**, as shown in Figure 2-17.

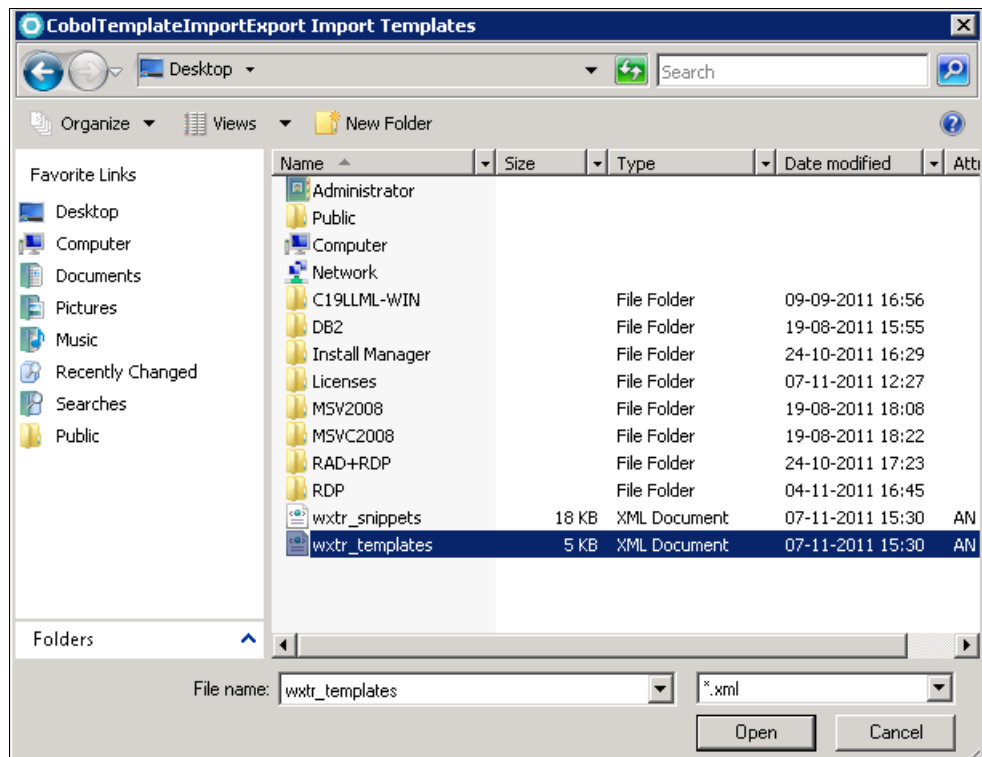


Figure 2-17 Selecting the WXTR-supplied COBOL program templates for importing

Important: If the wxtr_templates.xml file is corrupt, the importing process is likely to fail. If this failure occurs, an alternative is to copy the file in binary format directly to the system where RDp is installed. Do not edit the XML file before it is imported or copied.

- Click **OK** to close the Preferences window. If the XML file was successfully imported, a CICS COBOL program template entry is displayed in the File Templates window, as shown in Figure 2-18. The use of the template is explained in later sections of this chapter.

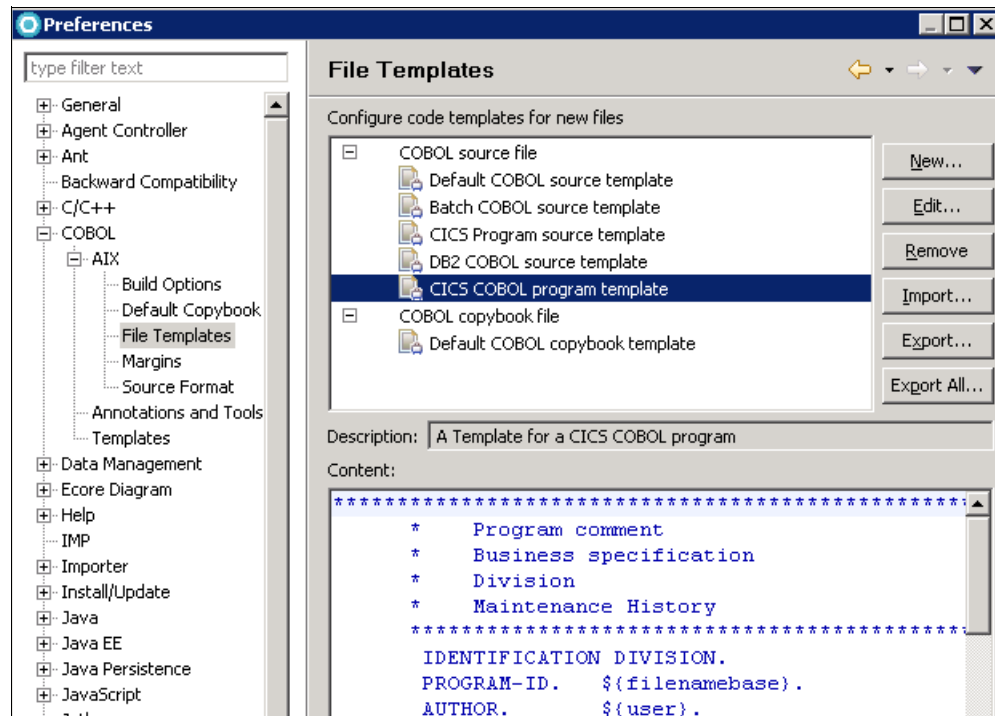


Figure 2-18 Imported CICS COBOL program template displayed in the File Templates window

WXTR-supplied snippets

WXTR supplies Java code snippets in RDP that can be used to create a connection to a WXTR container and to call COBOL programs (with or without application data). The application data can be in string or byte format. As Rational Application Developer and RDP are installed in the same IDE, Java application developers can import the WXTR-supplied snippets or the snippets that are imported to RDP.

Important: Rational Developer for Power Systems Software Version 8.0.3 ships with the WXTR snippets already included. Users directly use those snippets. There is no need to import the snippets again.

WXTR also provides makefile snippets to build CICS COBOL and COBOL projects that access DB2 resources. Complete the following steps to import the WXTR-supplied code snippets to RDP:

- Use binary mode to copy the *WXTR Install Directory/samples/RDP/wxtr_snippets.xml* file from the AIX machine where WXTR is installed to the Windows or Linux workstation where RDP is installed.
- Launch RDP.
- Open the AIX COBOL perspective by selecting **Window** → **Open Perspective** → **Other**, as shown in Figure 2-14 on page 38, and then select **AIX COBOL** from the list, as shown in Figure 2-15 on page 38.
- Open the Snippets view to import the code snippets by selecting **Window** → **Show View** → **Snippets**.

5. Right-click the **Snippets** view pane and select **Customize**, as shown in Figure 2-19.

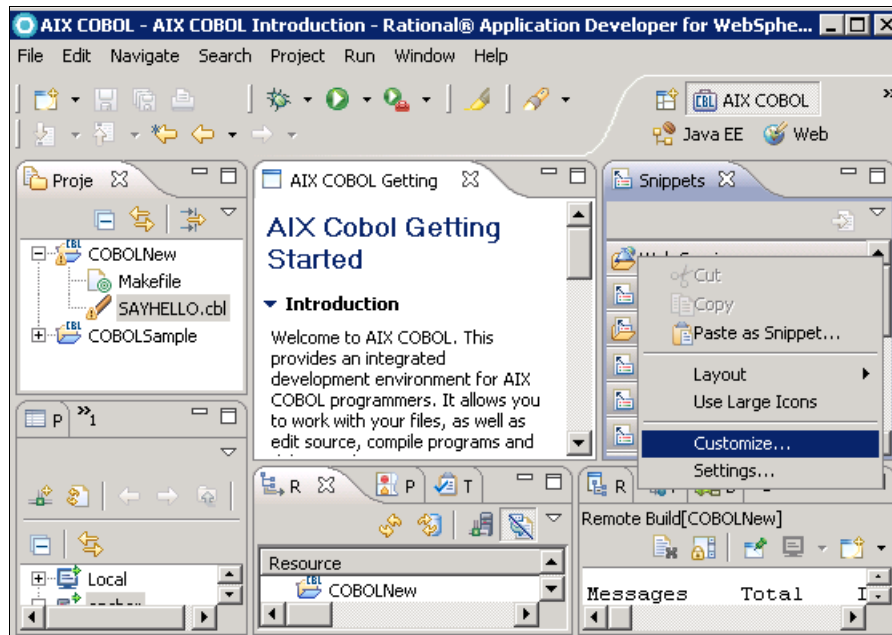


Figure 2-19 Opening snippets view to import the WXTR supplied snippets

6. In the Customize Palette window, click **Import**, as shown in Figure 2-20.

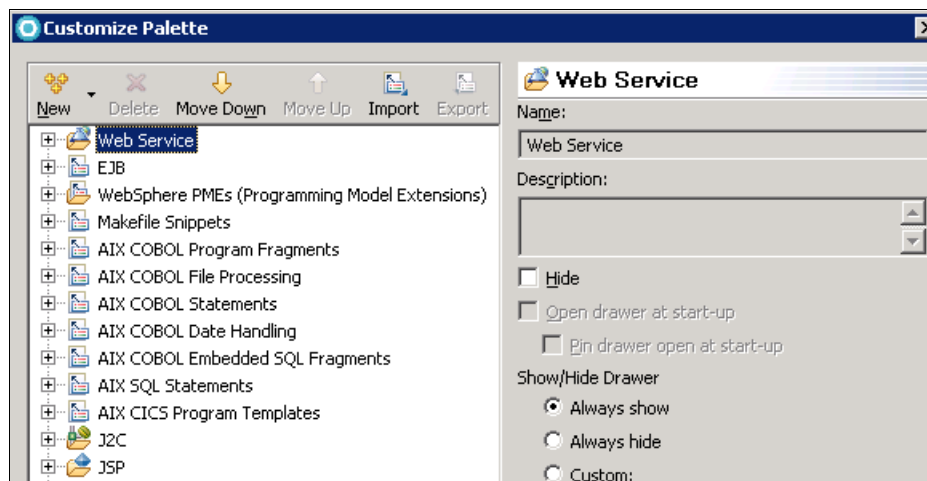


Figure 2-20 Customize Palette window to import the snippets

7. In the Import Snippets window, navigate to the directory that contains `wxtr_snippets.xml` and click **Open** and then click **OK**, as shown in Figure 2-21 on page 42.

Important: As with the XML file for the COBOL program templates, do not edit the snippets file before it is imported or copied. If the file is corrupt, the importing process may fail. If this occurs, a good alternative is to copy the file in binary format directly to the system where RDP is installed.

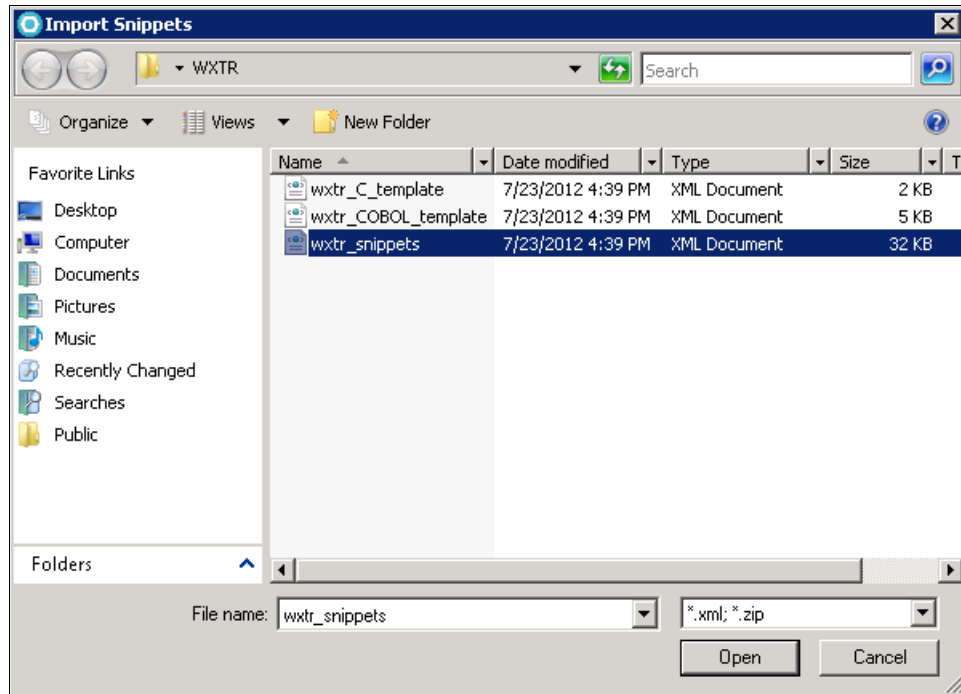


Figure 2-21 Choosing the WXTR supplied snippets file

If the XML file is successfully imported, the entry WebSphere eXtended Transaction Runtime (XTR) is created in the Snippets view, as shown in Figure 2-22.

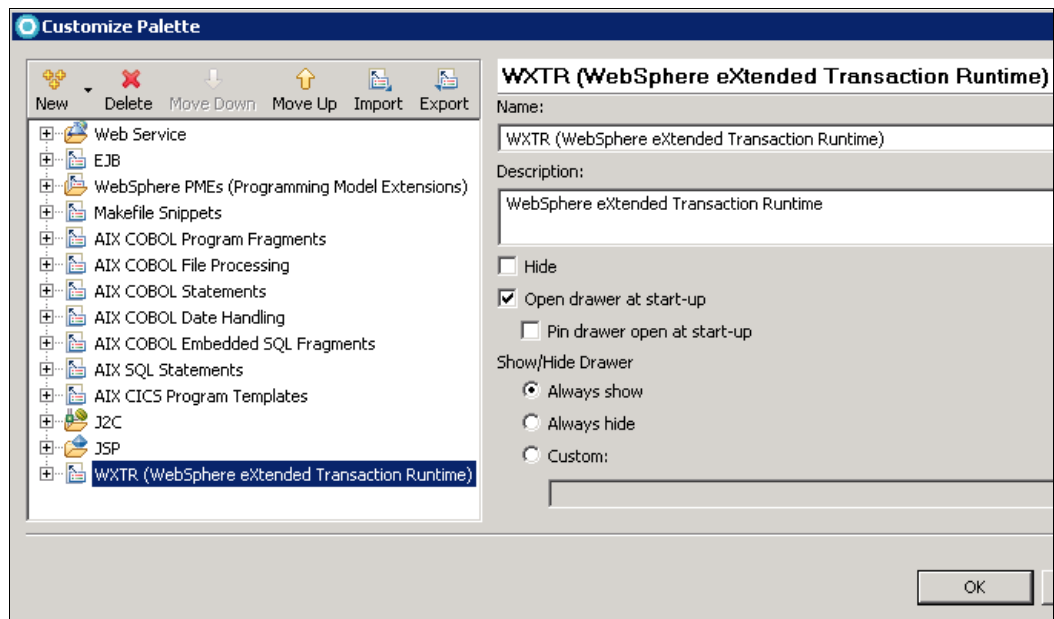


Figure 2-22 WXTR snippets added in the snippets view

WXTR supports the AIX operating system. For developing and deploying the CICS COBOL or C applications in WXTR container that is running on an AIX system, create a connection to the AIX system in Rational Developer for Power Systems Software.

Creating a connection to the remote AIX system

Rational Developer for Power Systems Software supplies a Remote System Explorer (RSE) component that must be installed on the AIX server to allow RDp to connect and work with remote systems. The component provides predefined plug-ins for browsing remote file systems, transferring files between hosts, performing remote searches, executing commands, and working with processes. By using the Remote System Explorer component, project-related files can be stored on the AIX server.

Complete the following steps to create a connection to the AIX server:

1. Launch RDp on your workstation.
2. The Remote System Explorer component on the AIX server will start a daemon process that uses 8050 as the default port number. Determine which process is running the daemon and on which port it is running by using the command sequence that is shown in Example 2-24.

Example 2-24 Command sequence to check the status of the RSE daemon process

```
# ps -u root -f | grep java.*ServerLauncher
root 12517462      1   0   Nov 07      -   0:07 /usr/java5/bin/java
-Xms64m -Xmx128m -Xss2m -DA_PLUGIN_PATH=/opt/IBM/RDPower/8.0/rse//
-DDSTORE_TRACING_ON=false org.eclipse.dstore.core.server.ServerLauncher 8050
```

Important: The daemon is running on port 8050 in this example. To configure a different port, edit the \$port variable of the daemon.pl script file that is located in the /opt/IBM/RDPower/8.0/rse/ directory. To affect the change, stop and restart the daemon process by using the **init 2** command.

3. Open the Remote System Explorer perspective by selecting **Window** → **Open Perspective** → **Remote System Explorer**, as shown in Figure 2-23.

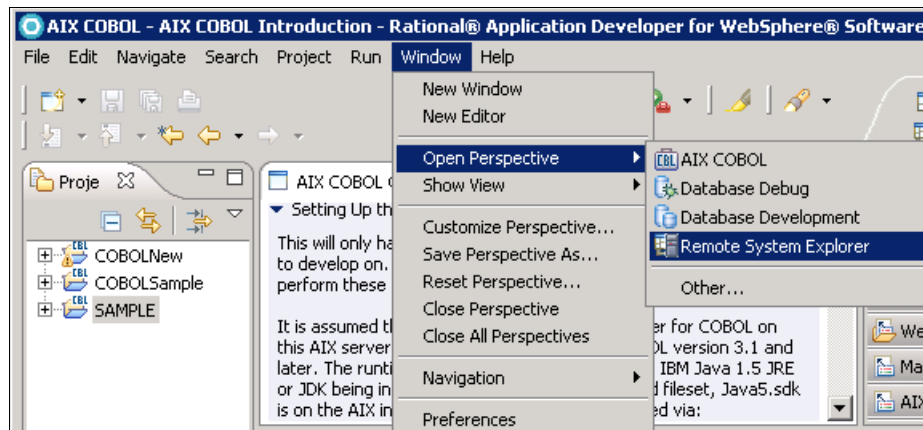


Figure 2-23 Opening Remote System Explorer perspective

4. To create a connection to the AIX server, select **File** → **New** → **Other**, as shown in Figure 2-24 on page 44.

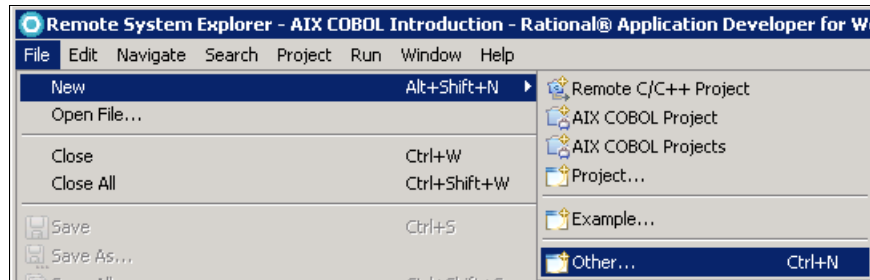


Figure 2-24 Creating new connection to the AIX server

5. In the Select a wizard window, enter **connection** as input to the Wizards attribute. Select the **Connection** wizard from the list of wizards and then click **Next**, as shown in Figure 2-25.

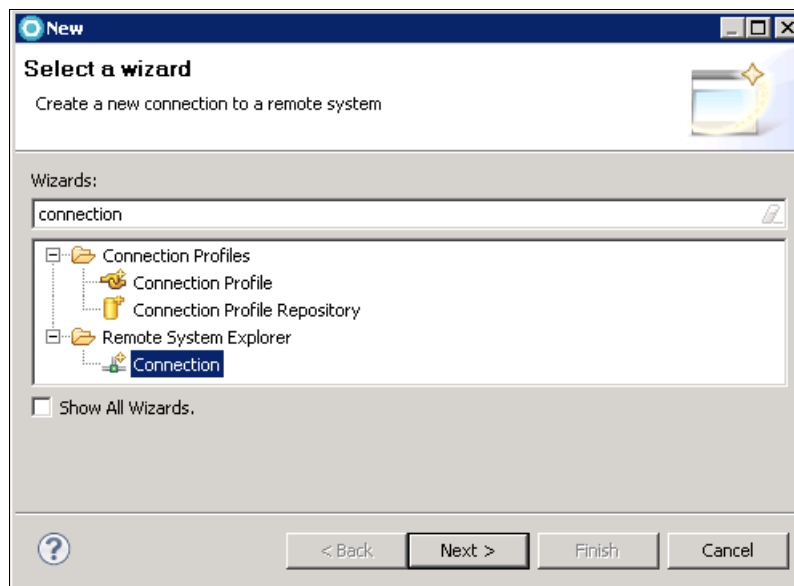


Figure 2-25 Selecting Connection wizard from the wizard list

6. In the New Connection window, select **AIX** as the remote server type and then click **Next**, as shown in Figure 2-26 on page 45.

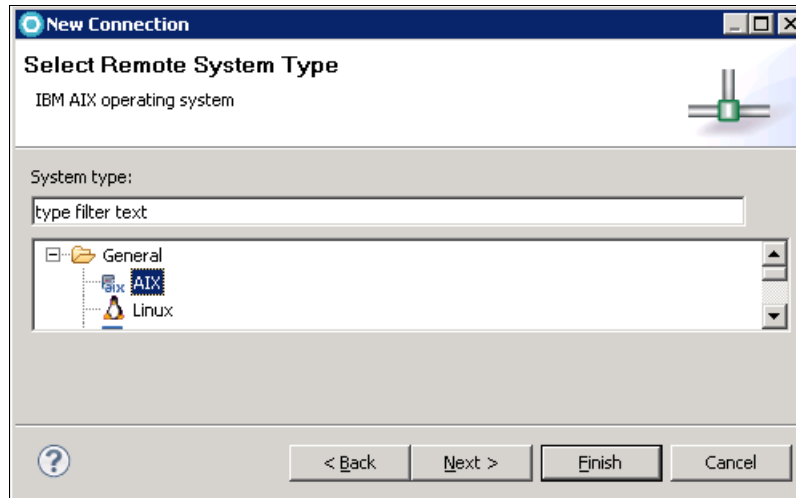


Figure 2-26 Operating systems list for creating the connection

7. In the New Connection window enter the name in the Host name field of the remote AIX server where WXTR is installed. Click **Next**, as shown in Figure 2-27.

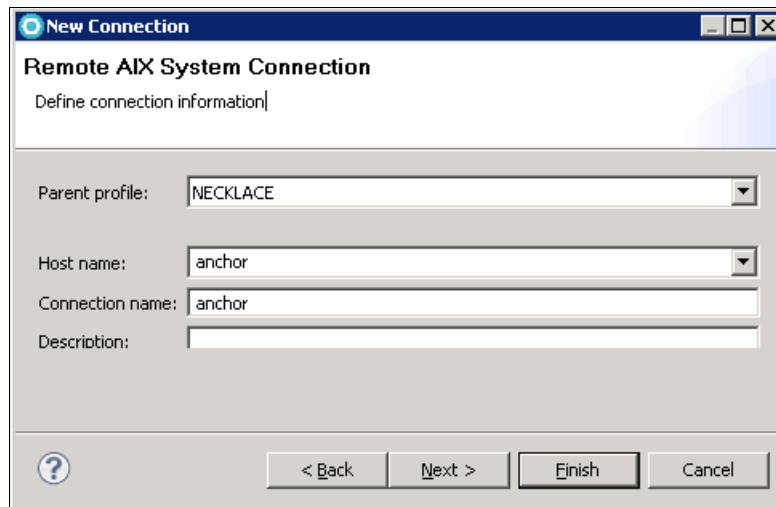


Figure 2-27 Specifying the remote AIX server details

8. Enter the port number as the Daemon Port attribute in the Connection Configuration section of New Connection window, then click **Finish**, as shown in Figure 2-28 on page 46. The port number can be determined by using the instructions that are provided in Step 2 on page 43.

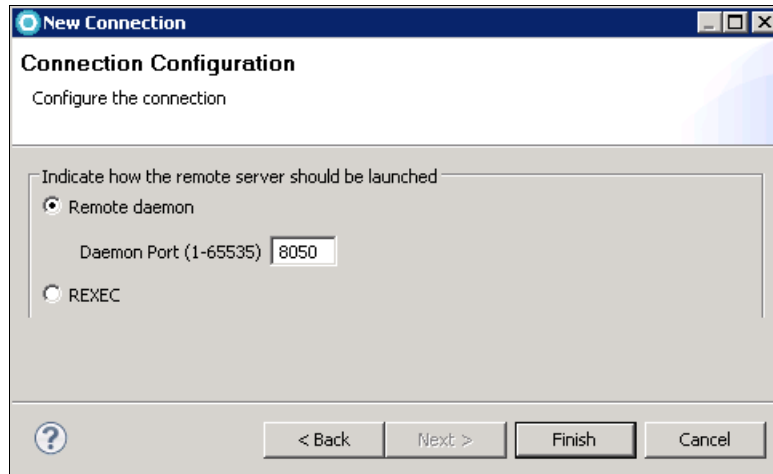


Figure 2-28 Configuring the remote AIX server connection

The new connection is created with the hostname that was specified in Step 7 on page 45, and is displayed in the list of remote systems, as shown in Figure 2-29.

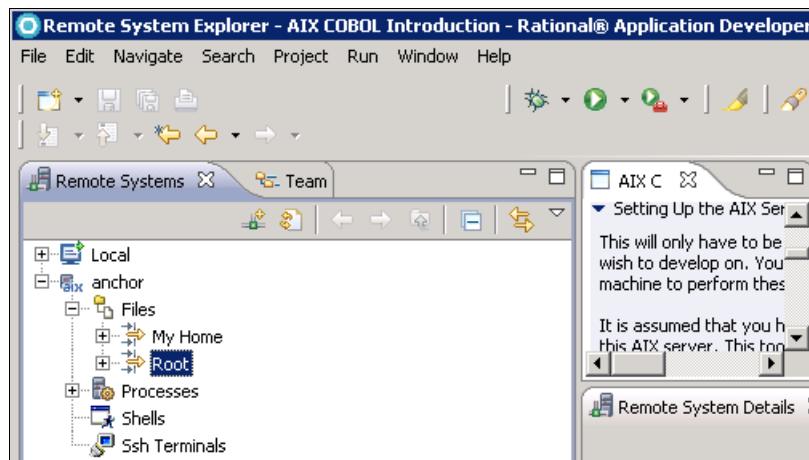


Figure 2-29 Select the Remote System connection entry from the list

9. Expand the hostname, AIX server name to which RDP connects, and select **Root**, as shown in Figure 2-29. Complete the following steps to enter the user credentials to access the remote system:
 - a. Enter the appropriate username and password to connect to the remote AIX server, as shown in Figure 2-30 on page 47.
 - b. Select the **Save user ID** and **Save password** options.
 - c. Click **OK**.
10. The connection is created. The hostname of the AIX server name and user credentials are saved in the Rational Developer for Power System Software.

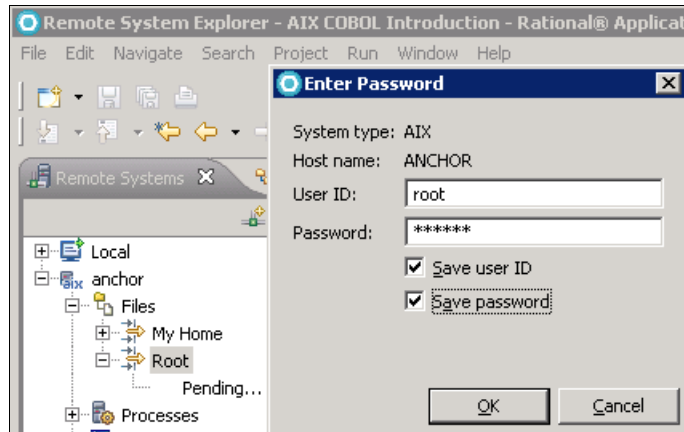


Figure 2-30 Specifying user credentials for the AIX connection

After the connection is created successfully, any folder on the remote system can be accessed by using the Remote System Explorer.

Developing COBOL applications

This section explains how to create COBOL applications for WXTR by using RDP with a focus on examples of COBOL applications that use CICS services. Creating an AIX COBOL project and importing existing CICS COBOL applications into WXTR also is described. Makefile creation for building the COBOL project also is described.

Creating a CICS COBOL project

Complete the following steps to create a CICS COBOL project by using RDP:

1. Launch RDP on your workstation.
2. Open the AIX COBOL perspective by selecting **Window** → **Open Perspective** → **Other** and then choosing **AIX COBOL** from the menu, as shown in Figure 2-14 on page 38 and Figure 2-15 on page 38.
3. Create an **AIX COBOL** project by selecting **File** → **New** → **AIX COBOL Project**, as shown in Figure 2-31. The AIX COBOL Project window opens.

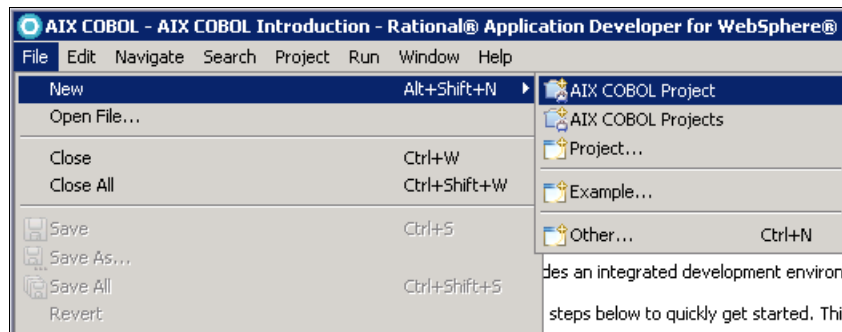


Figure 2-31 Creating an AIX COBOL project

4. Complete the following steps in the AIX COBOL Project window:
 - a. Enter the **Project Name** (for example, COBOLSample) as shown in Figure 2-32 on page 48.

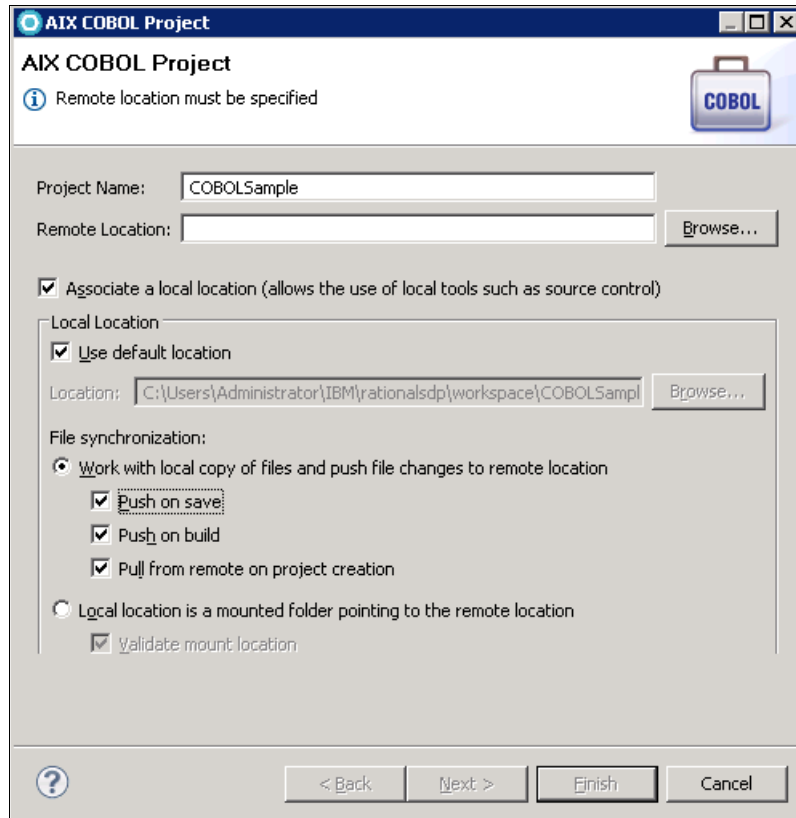


Figure 2-32 Specifying the project details for the new AIX COBOL project

- b. If a connection to the remote AIX server where the CICS COBOL programs will be placed is already created, enter the **Remote Location**.

You can browse to find the location, which is useful if you have not yet created the folder in which the programs will reside. You also can enter the path to the programs folder in the format (AIX connection name) path, where AIX connection name is the connection name and path is the directory path where the programs are to be located.

If you have not yet created a connection to the AIX server, do so now by following the steps in “Creating a connection to the remote AIX system” on page 43. Provide the location in the manner that is described here.

- c. In the File synchronization area, select the **Push on save**, **Push on build**, and **Pull from remote on project creation** options.
 - d. Click **Next**. The next panel in the AIX COBOL Project window opens in which you specify the build options.
5. Select the build options as shown in Figure 2-33 on page 49. Rational Developer for Power Systems Software uses the **make** command to build COBOL projects on AIX systems. There is no need to change the **build** command from its default setting (make) because WXTR also supports the use of the **make** command to build CICS COBOL applications.

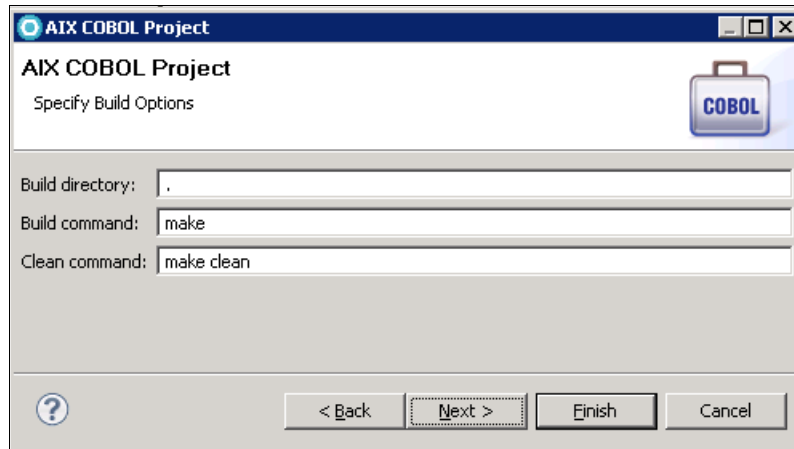


Figure 2-33 Selecting Build options for building the COBOL project

6. Click **Finish**. Your AIX COBOL project is created and is listed in the Project Explorer workspace.

Creating new COBOL program file

Rational Developer for Power Systems Software provides templates to create COBOL program files. Complete the following steps to create a COBOL program file:

1. Launch RDp on your workstation and open an AIX COBOL project, such as the one you created.
2. Right-click the COBOL project and select **New** → **AIX COBOL File**, as shown in Figure 2-34. The COBOL file window opens.

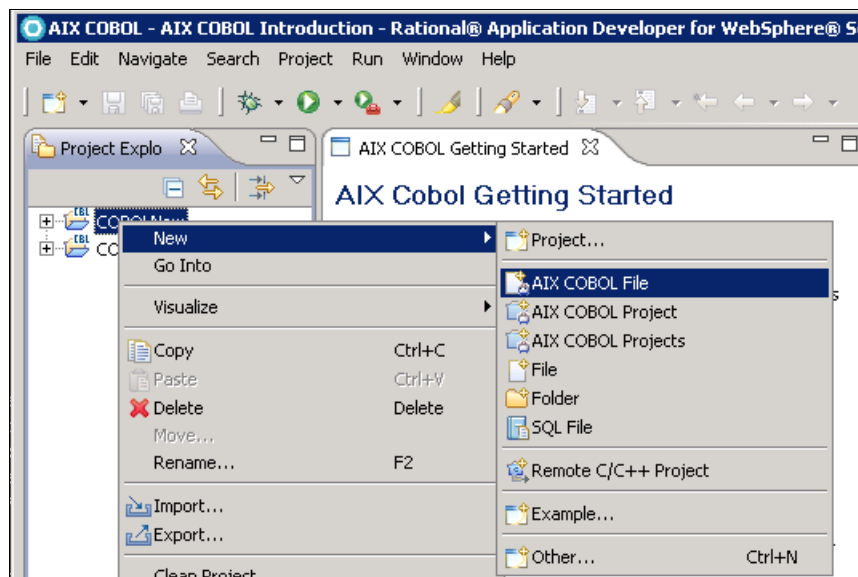


Figure 2-34 Creating new AIX COBOL programming file

3. In the COBOL file window, enter a name for the program in the File name field. Select the **Initialize the file content from a template** check box and choose **CICS COBOL program template** from the templates list, as shown in Figure 2-35 on page 50.

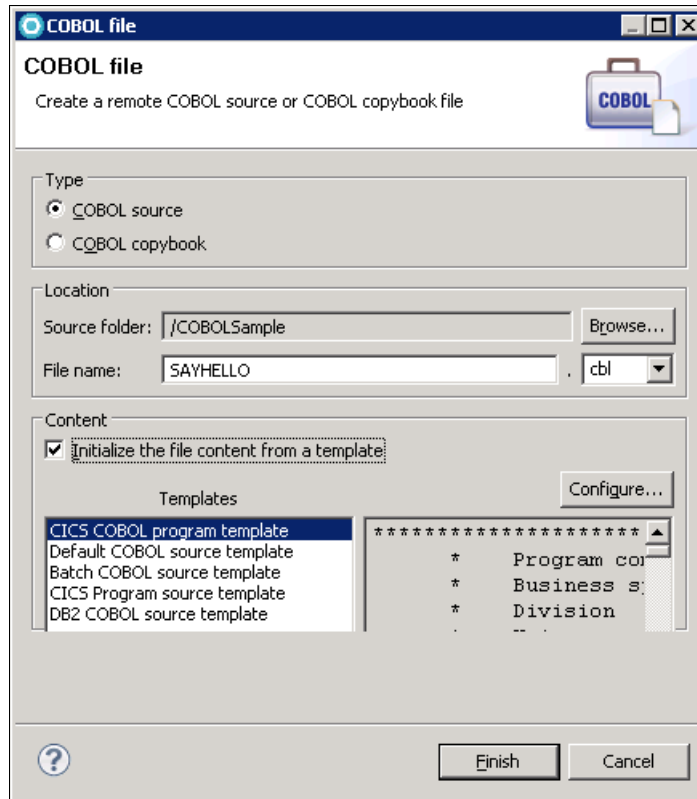


Figure 2-35 Creating new COBOL file with WXTR supplied template

Important: WXTR also supplies CICS COBOL program templates in the `wxtr_templates.xml` file. For more information about these templates and the procedure to import the file to RDp, see “WXTR-supplied templates” on page 37.

4. Click **Finish**. The new COBOL file is created with the selected template.

Important: If the Project → Build Automatically option is enabled by the user and the connection to the AIX server is not active, RDp throws build errors and you should check the connection details. Another potential failure that is related to this selection can occur if RDp tries to build the COBOL project with the `make` command. This attempt will fail because the makefile does not yet exist in the project. Ignore this error for now until the makefile is created.

Creating the makefile for building the COBOL project

WXTR also supports `make` command to build CICS COBOL projects that are created with Rational Developer for Power Systems Software.

Complete the following steps to create a makefile file in the AIX COBOL project:

1. Right-click the COBOL project and select **New** → **File**, as shown in Figure 2-36 on page 51.

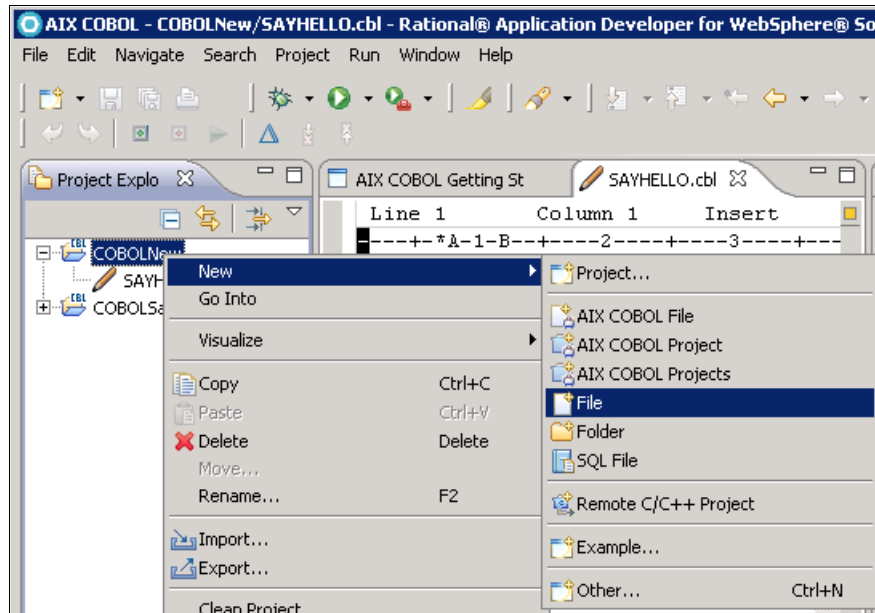


Figure 2-36 Creating new file in COBOL project

2. In the New File window, enter **Makefile** in the File name field (as shown in Figure 2-37) and click **Finish**. An empty file named Makefile is created. Import the WXTR-supplied makefile snippet into the file.

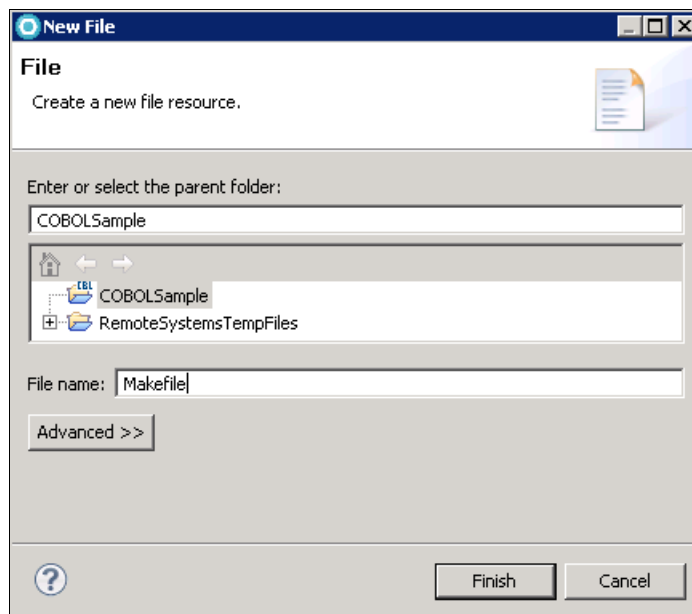


Figure 2-37 Entering the file name as Makefile

3. WXTR supplies makefile snippets. To import these snippets into the Rational Developer for Power Systems Software, see “WXTR-supplied snippets” on page 40.
4. To open the Makefile snippets from Rational Developer for Power Systems Software, open the snippets view by selecting **Window** → **Show View** → **Snippets** and then select **WXTR (WebSphere eXtended Transaction Runtime) snippets**, as shown in Figure 2-38 on page 52.

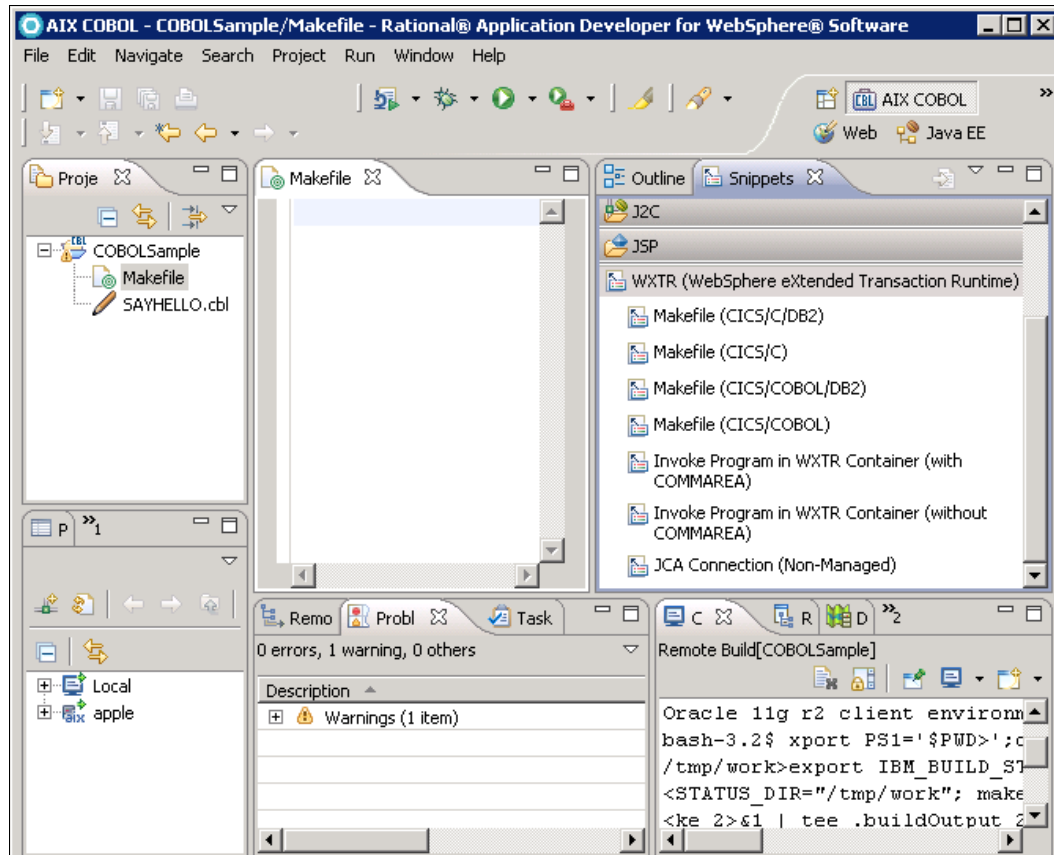


Figure 2-38 Selecting WXTR supplied snippets from Snippets view

5. If the CICS COBOL applications in the COBOL project use EXEC SQL statements to access data in DB2, choose the **Makefile(CICS/COBOL/DB2)** snippet. Otherwise, choose the **Makefile(CICS/COBOL)** snippet.

Double-click the snippet to select it. The **Insert Template:Makefile (CICS/COBOL)** window opens (as shown in Figure 2-39 on page 53) in which the variables can be modified.

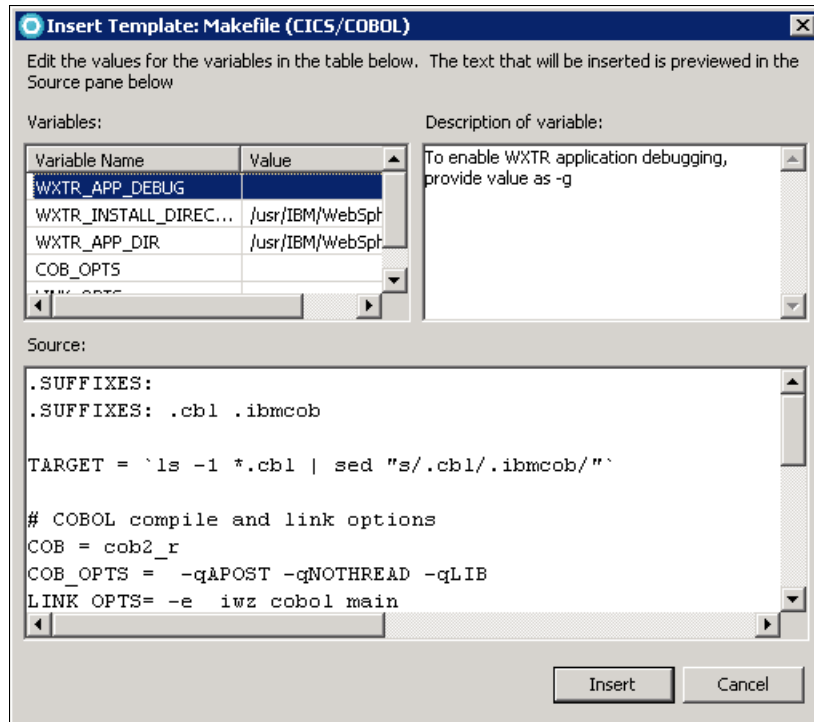


Figure 2-39 Editing WXTR supplied makefile template

6. Click **Insert** to insert the makefile snippet into the file.
7. Save the file by selecting **File** → **Save**. The makefile is now ready to use to build the COBOL project.

Importing COBOL programs into a project

Complete the following steps to import an existing COBOL application into an AIX COBOL project in RDp (these steps assume that the AIX COBOL project was created as described in “Creating a CICS COBOL project” on page 47):

1. Select **File** → **Import**.
2. In the Import window, select **Remote Systems**. Click **Remote file system**.
3. Click **Next**.
4. In the Remote file system window, click **Browse**.
5. Navigate to the location of the CICS COBOL programs and select the programs to be imported from the list that is provided, as shown in Figure 2-40 on page 54. Be sure to specify the AIX COBOL project in the Into folder field.

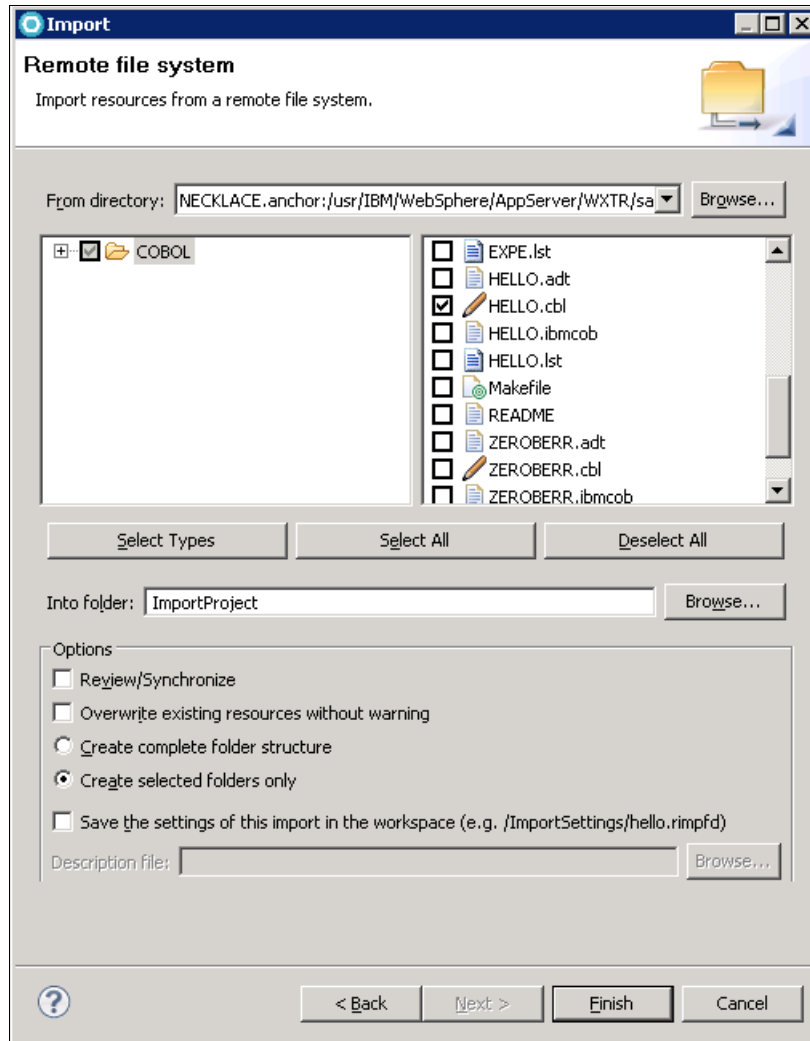


Figure 2-40 Import window to select the COBOL programs

6. Click **Finish** to import the selected files.
7. After the programs are imported, create the makefile to build the COBOL project. For more information about creating the makefile, see “Creating the makefile for building the COBOL project” on page 50.

Editing COBOL programs with Remote Systems LPEX Editor

With Rational Developer for Power Systems Software, the COBOL editing process is similar to that of Rational Developer for System z. The developer edits COBOL source code by using the Remote Systems LPEX Editor.

The Remote Systems LPEX Editor provides COBOL language parsing and includes the following features:

- ▶ Content Assist to insert COBOL language elements, user-defined words, and templates into your code, including embedded CICS and SQL, as shown in Figure 2-41 on page 56.
- ▶ The ability to compose COBOL programs from pre-existing COBOL file templates, content-assist templates, and snippets. You can also add your own templates and snippets and distribute them among your team.
- ▶ Flyover displays of variable declarations.

- ▶ The ability to navigate to data declarations and paragraphs by using hot keys and hyperlinks.
- ▶ Definitions inside copybooks that can be discovered and participate in the source navigation and content assist proposals.
- ▶ Intelligent refactoring for renaming variables, which removes unnecessary COBOL keywords and extracts selected lines into paragraphs.
- ▶ A call hierarchy that can be displayed in a Perform Hierarchy view and used for navigating the source.
- ▶ A filterable outline view that shows the structure of the COBOL code and can be used to navigate through it.

Adding business logic to the COBOL programs

WXTR provides the following CICS services for CICS COBOL application development:

- ▶ Business logic services:
 - Program execution
 - Synchronization
 - Storage
 - Logical Units of Work
 - Configuration
 - Timers
- ▶ Data services:
 - File services that allow DB2 files as VSAM files:
 - KSDS
 - ESDS
 - RRDS
 - Queue services:
 - Transient data queue (TDQ)
 - Temporary storage queue (TSQ)
 - Journal services
 - Relational database services

WXTR supports a set of CICS APIs for each of these services. For a list of the supported CICS APIs, see the *Application Programming Interfaces* section of the WXTR Information Center at this website:

<http://publib.boulder.ibm.com/infocenter/wxtrform/v2r1/index.jsp>

Application programmers can use the CICS APIs to develop or enhance CICS COBOL programs by using RDp features, such as highlighting COBOL and embedded CICS and SQL verbs, content assist, snippets, templates, API errors, and warnings.

Rational Developer for Power Systems Software provides a Content Assist feature that can highlight the COBOL verbs and the embedded CICS and SQL verbs in the application, as shown in Figure 2-41 on page 56.

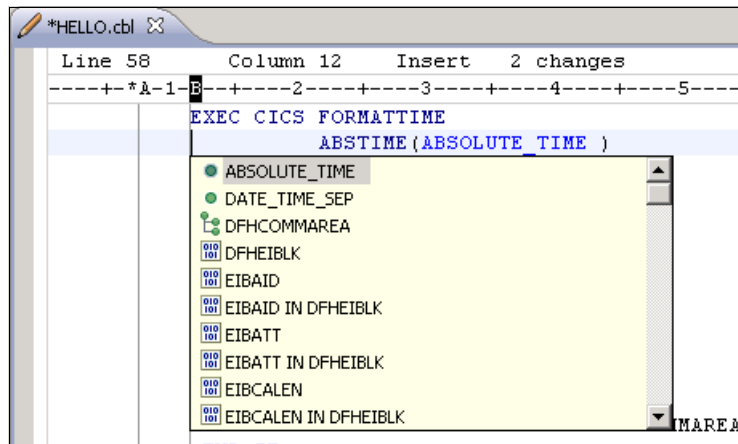


Figure 2-41 Content assist feature for CICS APIs

To use Content Assist, open a COBOL file in the COBOL Editor. As you are writing the code, press **Ctrl + Spacebar** or click **Edit → Content Assist** to select from the displayed options.

Figure 2-41 shows the content assist for CICS API `FORMATTIME`. Double-click an option to see a list of the user-defined variables (see Figure 2-42) that can be included in applications as required.

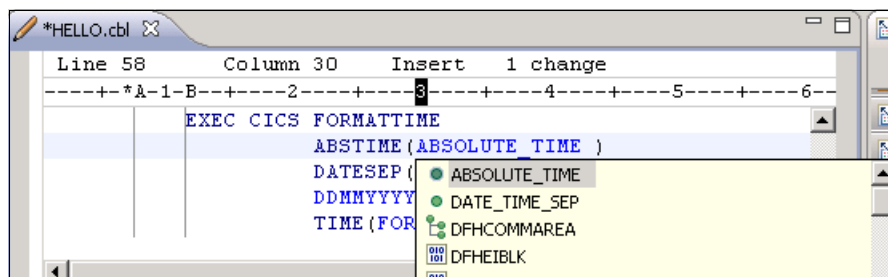


Figure 2-42 Content assist feature to display user-defined variables

The variable names also are highlighted in the editor, as shown in Figure 2-42. If you make any errors when you are entering the CICS API options, the editor suggests the correct option, as shown in Figure 2-43.

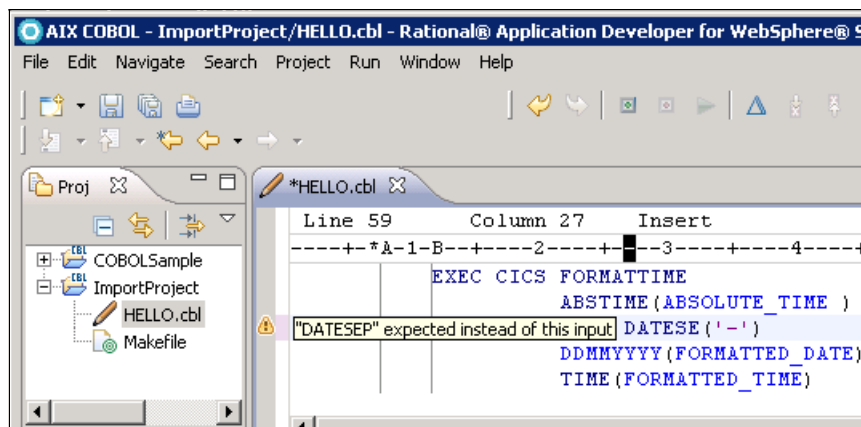


Figure 2-43 Suggesting the correct options for typographical errors

If any user-defined variable or option in the CICS API is invalid, error messages are displayed in the editor, as shown in Figure 2-44.

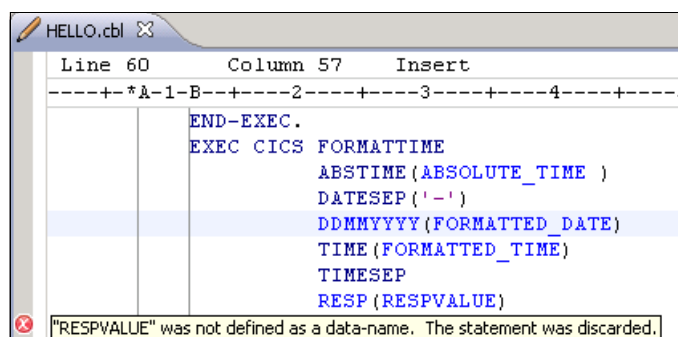


Figure 2-44 Error message for undefined variables

For more information about these Content Assist features, see the Rational Developer for Power Systems Software Information Center at this website:

<http://publib.boulder.ibm.com/infocenter/iadthelp/v8r0/topic/com.ibm.etools.unix.cobol.projects.doc/topics/rAIXCOBOLSharedEditor.html>

If the CICS COBOL application accesses files on DB2 or works with a TDQ, add the respective resource entry to the WXTR container by using the wsadmin tool. For more information about adding resource definitions to a WXTR container, see section 2.3.2, “Using the command line” on page 32.

Building COBOL project

For building CICS COBOL project, you must create the makefile as described in “Creating the makefile for building the COBOL project” on page 50. Use either of the following approaches to build the project:

- ▶ Right-click the project and select **Build Project**.
- ▶ Select **Project** → **Build Automatically**.

Testing COBOL applications

To test CICS COBOL applications, develop a Java EE application by using the JCA or SCA interface that is provided by WXTR to invoke the COBOL application then run the Java EE application. For more information about this process, see 2.4.4, “Accessing CICS COBOL through JCA and SCA” on page 68.

You can verify CICS COBOL applications by running them through the samples portal that are provided by WXTR by completing the following steps:

1. Open the WXTR samples portal by entering the application URL
<http://hostname:port/WXTRSample/> in a browser. In this example URL, hostname is the name or IP address of the machine where WebSphere Application Server is running, port is the WebSphere Application Server HTTP transport port, and WXTRSample is the context root that is used to access the deployed WXTRSample web application.
2. In the pane on the left side of the page, select **Miscellaneous** → **Invoke Program**.
3. In the pane on the right side of the page, enter the program name and application data in the Input COMMAREA field.
4. Click **Invoke**.

If the program runs successfully and the application returns data, you see the output data in the output COMMAREA field.

Log files

Messages that are received while CICS applications are running in WXTR append the following log files:

- ▶ **WXTR container console file:** This file, `console.nnnnnn` (where `nnnnnn` represents a number that is incremented every time a new console file is opened), is in the `\LogsPath` directory. All CICS related runtime information messages, warnings, and error messages are logged in this file. COBOL runtime errors also are logged here.

Retrieve the `\LogsPath` directory by using the `wsadmin` command by querying for the `LogsPath` property in the RD resource class.
- ▶ **WXTR container output:** This file, `console.msg`, contains output messages from COBOL applications (such as from the `DISPLAY` statement) and is located in the `\LogsPath` directory. The `console.msg` files are created on every restart of the WXTR container.
- ▶ **SystemOut:** This is a standard JVM output file for WebSphere Application Server that is in `${SERVER_LOG_ROOT}`.
- ▶ **SystemErr:** This is a standard JVM error file for WebSphere Application Server in `${SERVER_LOG_ROOT}`. COBOL application runtime errors are logged as exceptions in this file.

These log files can be accessed directly or, if you are using the WXTR-supplied samples portal, can be accessed from the Logs section on the left side of the portal window.

Debugging COBOL applications

WXTR brings a unified experience of debugging both Java EE and CICS COBOL applications in a single IDE. As you debug a Java EE program by stepping through its execution, you also can step into a CICS COBOL application in the same IDE. To debug CICS COBOL applications that are running in WXTR, use IBM Distributed Debugger, which uses a client/server design model.

Setting the debug mode



When you are using IBM Distributed Debugger, enable debugging of CICS COBOL applications by completing the following steps:

1. Compile CICS COBOL applications by using the `-g` compiler option.
2. Set the environment that is shown in Example 2-25 in the corresponding WXTR `profile_root/config/cells/cellName/nodes/nodeName/servers/serverName/serverLvlWxtr.properties` file. The default library path location of IBM Distributed Debugger is `/usr/idebug/engine/lib`. The IP address is the debug UI IP address or the machine name. The port number is the port on which the debug UI daemon is listening.

Example 2-25 Environment variables to be set for CICS COBOL application debugging

```
CICS_IDEBUG_LIBPATH=<Library path of IBM Distributed Debugger>
CCW_DEBUG_IPPORT=<IP address>:<Port number>
```

Complete the following steps to obtain the IP address on the IBM Rational Developer for Power Systems window:

- a. Click **Window** → **Show view** → **Other** → **Debug** and then select **Debug** to open the Debug view.
- b. Click the Debug UI daemon icon. The status of the daemon is indicated by the icon. If the daemon is listening, the icon is shown as: . If the daemon is not listening, the icon is shown as: .
- c. Select the option **Get Workstation IP**, as shown in Figure 2-45 on page 59.

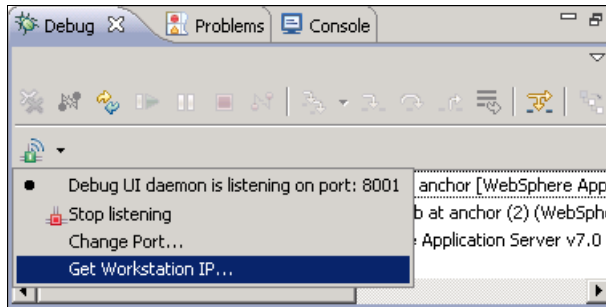


Figure 2-45 Getting Workstation IP and port details for enabling debugging

- d. To obtain the Port number, select the **Debug UI daemon is listening on port:** option, as shown in Figure 2-45. The listening port is displayed.
- e. To change the port number on which the Debug UI daemon is listening, select the **Change Port** option.

Importing WXTRSample.war to Rational Application Developer

WXTR supplies a samples portal that is used for invoking different COBOL sample applications. Sample portal can be launched by importing WXTRSample.war and starting the WXTRSample application. Complete the following steps to import the WXTRSample.war file to Rational Application Developer:





1. Start the Rational Application Developer program on a workstation that is running Windows.
2. Define a new server for WebSphere Application Server that is running on AIX. For more information, see the Rational Application Developer Information Center at this website:
http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.servertools.doc/topics/twcertins_v6.html
3. Use FTP to transfer the following files in binary mode from the AIX system to the Windows workstation (<WXTR Install Directory> is the directory in which WXTR is installed):
 - WXTR Install Directory /samples/J2EE/WXTRSample.war
 - WXTR Install Directory /samples/J2EE/SMDemo.war
 - WXTR Install Directory /lib/WXTRConnector.jar
 - WXTR Install Directory /lib/wxtrsca.jar
4. Import the WXTRSample.war file by completing the following steps:
 - a. Select **File** → **Import** → **WAR file**.
 - b. In the **WAR import** window, enter the path to the location of the WXTRSample.war file that was copied in the previous step.
 - c. Enter the values for **Web Project** and **EAR project name**.
 - d. Click **Next**.
 - e. In the **WAR Import:Web libraries** window, select the **wxtrsca.jar** check box.
 - f. Click **Finish**.
5. Complete the following steps to add the external JARs to the project:
 - a. In the **Enterprise Explorer** window, right-click **WXTRSample project** and select **Properties**.
 - b. In the left pane of the Properties window, select **Java Build Path**.
 - c. In the right pane of the Properties window, select the **Libraries** tab.

- d. On the Libraries tab, click **Add External JARs** and enter the path to the location of the WXRConnector.jar file that was copied in Step 3 on page 59.
- e. Click **OK**.
6. Repeat steps 4 and 5 for the SMDemo.war file.
7. To include the wxtrsc.jar file in the Java build path of Rational Application Developer, follow the procedures in step 5 on page 59 and choose wxtrsc.jar in **Add External JARs** step.
8. The WXTR-supplied samples portal also supports SCA interfaces samples. To use SCA-based samples, right-click **WXTRSample project** to choose **Properties** → **Project Facets**, and then select the following check boxes:
 - Service Component Architecture (SCA)
 - WebSphere 7.0 Feature Pack for SCA
9. Import the wxtrsc.jar file as an SCA archive file into the **WXTRSample project** by using the Rational Application Developer import facility.

Important: Do not clean the project. If you clean the project, some of the imported files might be removed. To import the wxtrsc.jar file again as an SCA archive file into the WXTRSample project, use the Rational Application Developer import facility.

2.4.2 Debugging CICS COBOL applications

Complete the following steps to debug CICS COBOL applications:

1. Launch IBM Rational Developer for Power Systems Software on your workstation.
2. Open the Debug perspective and ensure that the Debug UI daemon is listening.
3. Note the state of the daemon icon in the Debug view. If the daemon is listening for debug connections, the Debug UI daemon icon is shown as: . If the daemon is not listening, the Debug UI daemon icon is shown as: .
4. If the debug daemon is not listening, start the daemon by using one of the following options:
 - Click the Debug UI daemon icon . It will change to green  to indicate that the daemon has started.
 - From the list of daemons, select **Start listening on port: <port number>** from the menu. Port number is the port on which the debug UI daemon is listening.
5. Confirm that you completed the steps in “Setting the debug mode” on page 58 and ensure that WebSphere Application Server test environment is restarted in Debug mode.
6. Set breakpoints in your Java EE application to the statements that you want to debug, as shown in Figure 2-46.

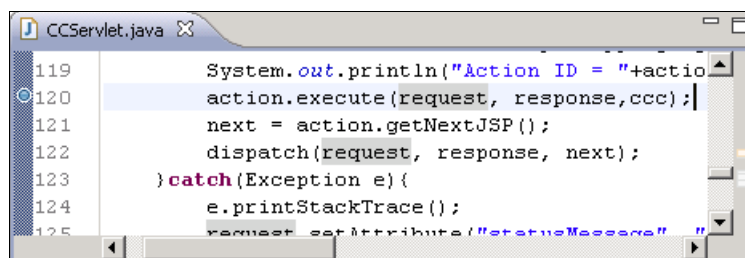


Figure 2-46 Adding breakpoints in a JAVA EE application

7. Select the AIX COBOL perspective as shown in Figure 2-14 on page 38 and Figure 2-15 on page 38.
8. Open the CICS COBOL source that you want to debug and set breakpoints. To add or remove a breakpoint, right-click any executable statement and select **Add/Remove Breakpoint**, as shown in Figure 2-47.

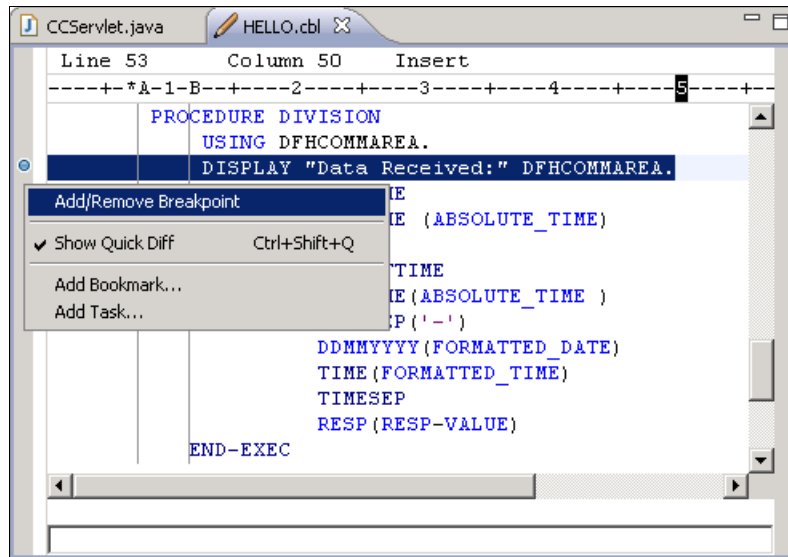





Figure 2-47 Adding breakpoints to a CICS COBOL application

9. In the Enterprise Explorer window, right-click the Java EE project that you want to debug. Select **Debug As** and then click **Debug on Server**.
10. In the **Debug on Server** window, choose the server that is defined for your WebSphere Application Server on AIX.
11. Click **Finish**. After the server is started in debug mode, the tool suspends running the program where the breakpoint is set. You can perform Step Into , Step Over , Step Return , and other debug operations through the Debug view.

Complete the following steps to debug a CICS COBOL application by using the WXTR-supplied samples portal:

1. Complete the process described in “Importing WXTRSample.war to Rational Application Developer” on page 59.
2. From the Enterprise Explorer window, right-click **WXTRSample** and repeat steps 9 and 10 from the previous process about debugging CICS COBOL applications. The **WXTRSample** starts, as shown in Figure 2-48 on page 62.

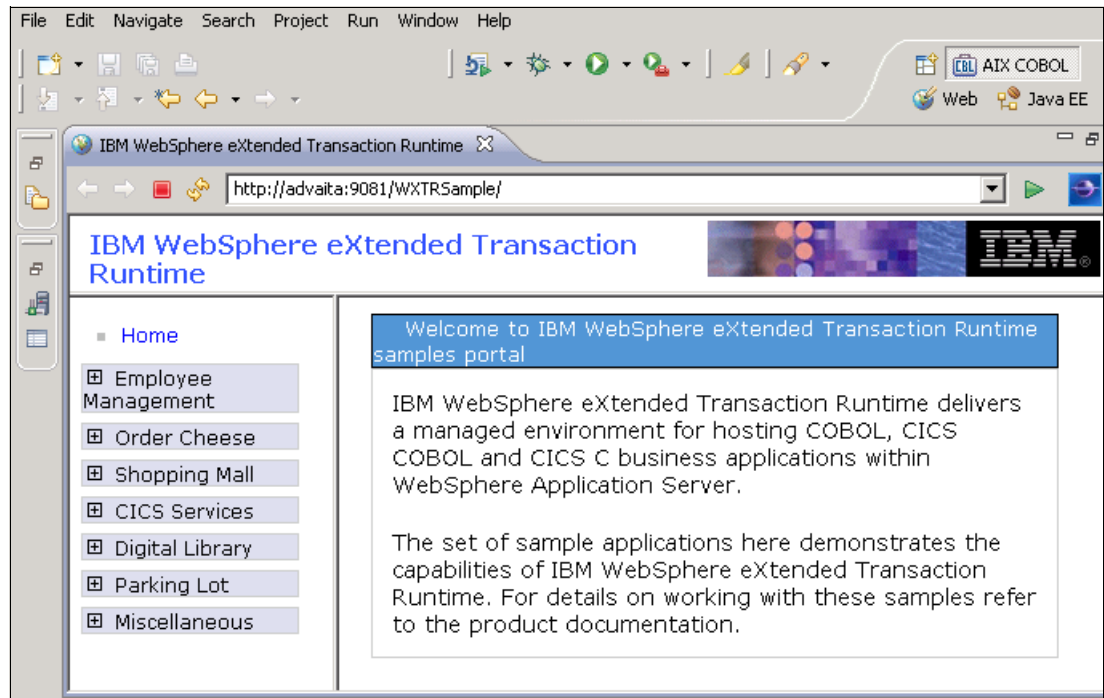


Figure 2-48 WXTR samples portal for debugging

3. In the left pane of the samples portal window, click **Miscellaneous** → **Invoke Program**. The Invoke Program window opens in the right pane, as shown in Figure 2-49 on page 63.
4. Provide the following information:
 - a. Enter the CICS COBOL application name in the **Program Name** field.
 - b. If the application requires any input data, enter the data in the **Input COMMAREA** field. This field accepts data in string format. If the application data is sent incorrectly, the WXTR container might throw an exception.
5. Click **Invoke**.

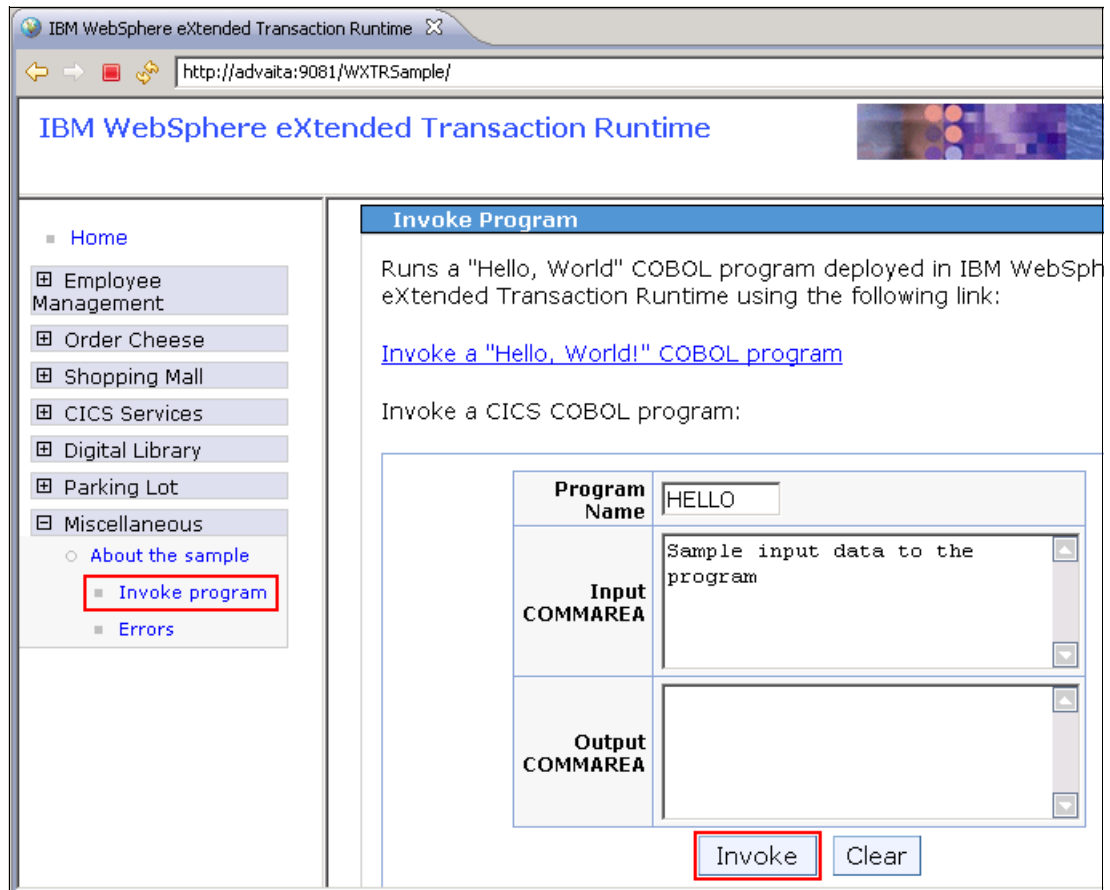








Figure 2-49 Samples portal to invoke the COBOL program

6. If any break points were set in the Java EE application, the control will stop there. You can perform Step Into  (F5), Step Over  (F6), or Step Return  (F7) to debug the Java EE code. If the cursor moves to a variable in the code, the runtime values for the variables are displayed, as shown in Figure 2-50 on page 64.
7. When the control reaches a call to a COBOL program in a Java application, the control automatically goes to the COBOL program. Use the Step Into  (F5), Step Over  (F6), or Step Return  (F7) controls to debug the COBOL application program.
8. The arguments to the CICS APIs and the variable values in the COBOL programs can be checked, as shown in the Figure 2-51 on page 64.

Important: Whenever a CICS COBOL program is executed in debug mode and the debug UI daemon is listening, a message window is shown that includes the statement, "The debugger has detached from the program."

For more debugging options, see the Rational Developer for Power Systems Software Information Center at this website:

<http://publib.boulder.ibm.com/infocenter/iadthelp/v8r0/topic/org.eclipse.jdt.doc.user/reference/views/debug/ref-terminateall.htm>

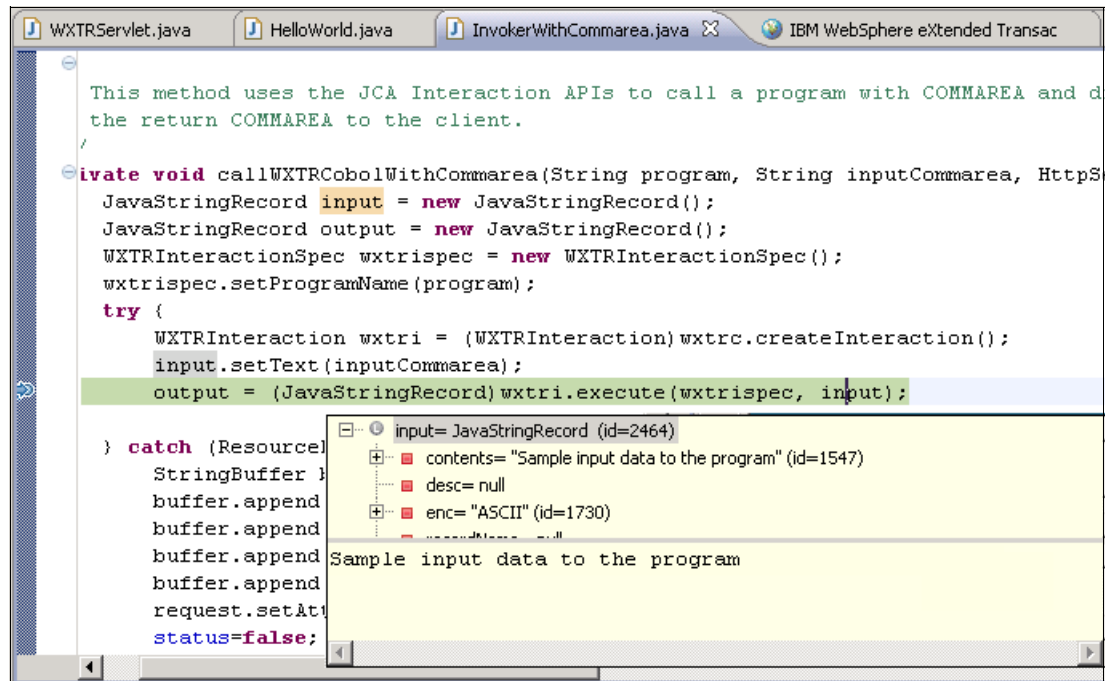


Figure 2-50 Java EE application debugging

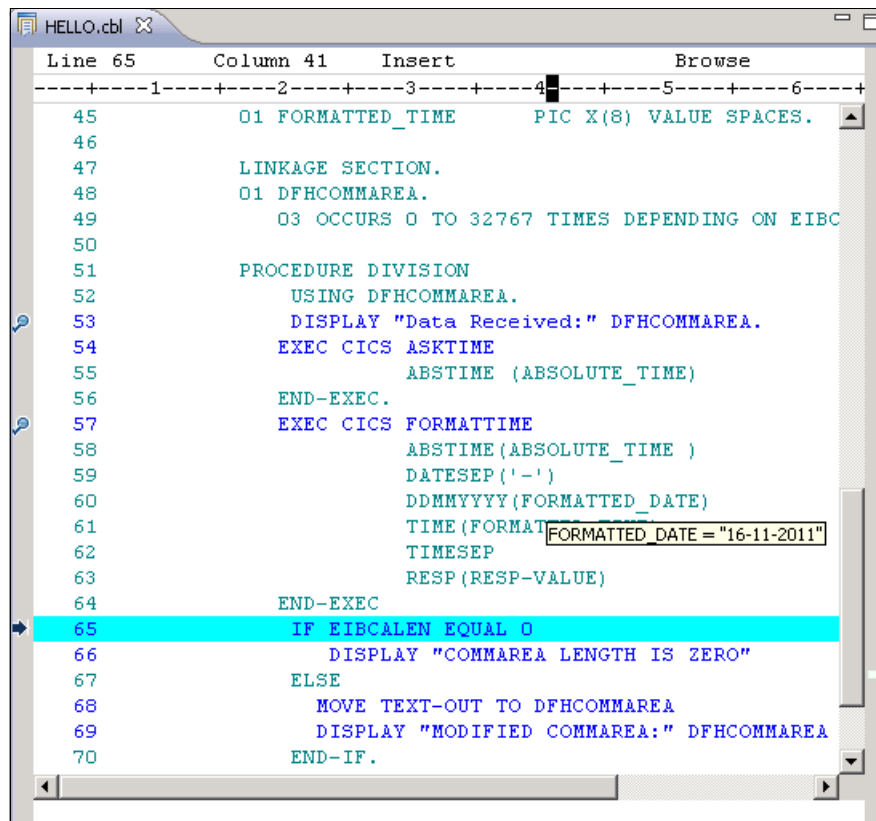


Figure 2-51 COBOL application debugging with rational developer for power systems software

2.4.3 Creating binding classes

Rational Application Developer for WebSphere Software provides the COBOL Importer, a component of the Java data binding tool. The COBOL Importer uses a data type transformation process to import existing COBOL programs to the binding tool. It also generates metadata information about the COBOL data structures, such as a COMMAREA in a CICS COBOL program. The COBOL Importer maps the data types in the source file to corresponding data types that can be accessed in a Java application.

The J2C Java Data Binding wizard, called *CICS/IMS Java Data Binding*, helps you to create Java classes that represent the input and output messages of a CICS transaction. These messages correspond to the COBOL data structures of a CICS COBOL program. The resulting specialized Java classes are called *data binding classes* and are used as input or output types for invoking CICS COBOL programs.

After you create the data binding classes, you can use the J2C wizard to create a Java bean that contains a method that uses the binding classes.

Complete the following steps to create Java classes from existing COBOL data structures by using the J2C Java Data Binding wizard:

1. Launch Rational Application Developer on your workstation.
2. Select **Window** → **Open Perspective** → **Other** and then choose **Java EE** from the Open Perspective window. The **Java EE** perspective is launched.
3. Select **File** → **New** → **Other** to open the Select a wizard window. Expand **J2C** and select **CICS/IMS Java Data Binding**.
4. Click **Next**. The Data Import window opens.
5. In the Data Import window, select **COBOL to Java** from the Choose mapping list.
6. Click **Browse** next to the COBOL file option to find and select the COBOL source file or COBOL copybook file.
7. Click **Next**. The sample COBOL copybook `RRDSCOPY.cpy` is shown in the COBOL file area, as shown in Figure 2-52. The copy book contents are defined in Example 2-26 on page 66.

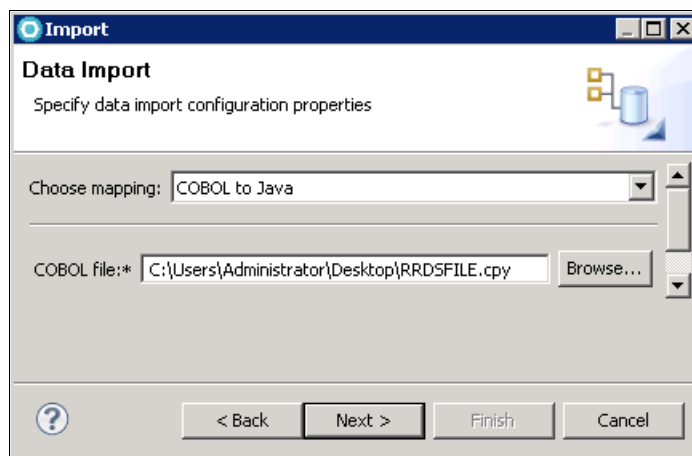


Figure 2-52 Selecting COBOL copybook for creating the data binding classes

Example 2-26 Sample COBOL copybook to create the binding

```
01 BOOK-ENTRY.  
    05 BOOK-REC.  
        10 STAT          PIC X.  
        10 BOOKNAME      PIC X(100).  
        10 BAUTHOR       PIC X(50).  
        10 PUBLISHER     PIC X(50).  
        10 CATEGORY      PIC X(30).  
        10 DATEOP        PIC 9(10).  
        10 COST          PIC 9(4).  
        10 PAGES         PIC 9(4).  
    05 BOOK-ID.  
        10 RRN-NO        PIC 9(4).
```

8. In the Importer window, select **AIX** as the Platform and **ISO-8859-1** as the Code page, as shown in Figure 2-53. Click **Query** and select a COBOL data structure from the list to be converted into a corresponding Java class.
9. Click **Next**.

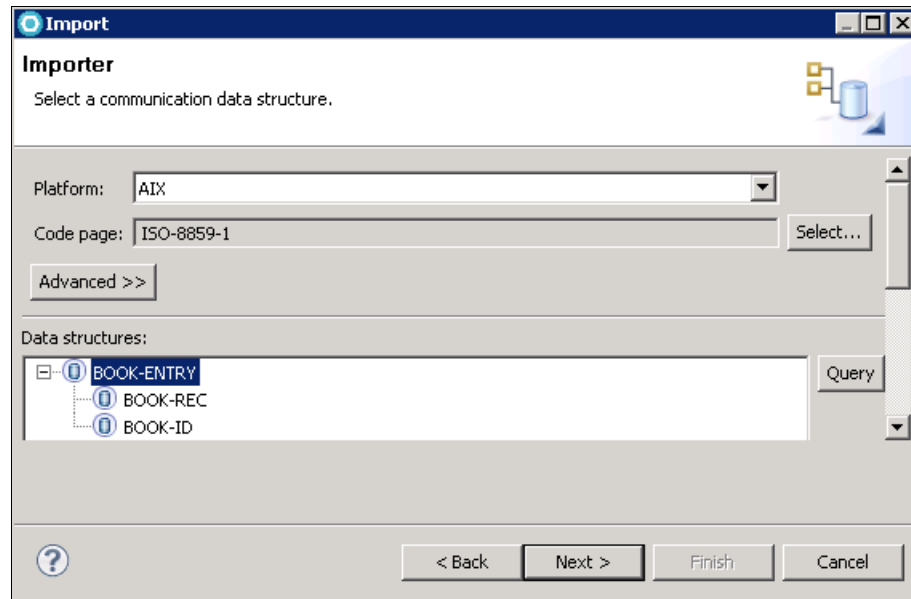


Figure 2-53 Selecting the Data structures from the COBOL copybook

10. In the Saving properties window, enter the **Project Name** or click **Browse** to find and select it. Specify the **Java package name** as the Package Name or click **Browse** to select it, as shown in Figure 2-54 on page 67.

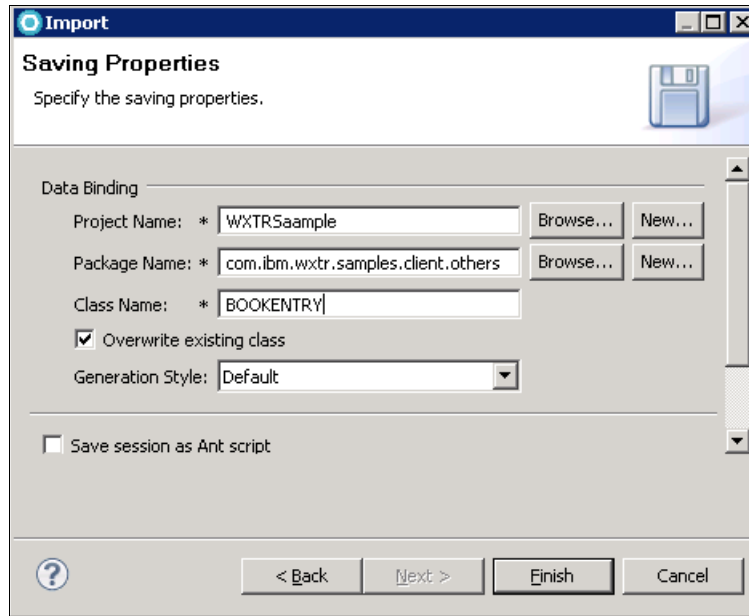


Figure 2-54 Selecting the Java project details for creating the binding classes

11. Specify a **Java class name** or accept the default value that is provided in the Class Name field.
12. Click **Finish**. The tool generates the Java class (see Figure 2-55) under the specified Java project. The appropriate setter and getter methods for all the variables in the COBOL data structure that are selected from the COBOL copybook are included. The Java client program now can instantiate the generated Java class and use it to construct and send COMMAREA data buffer.

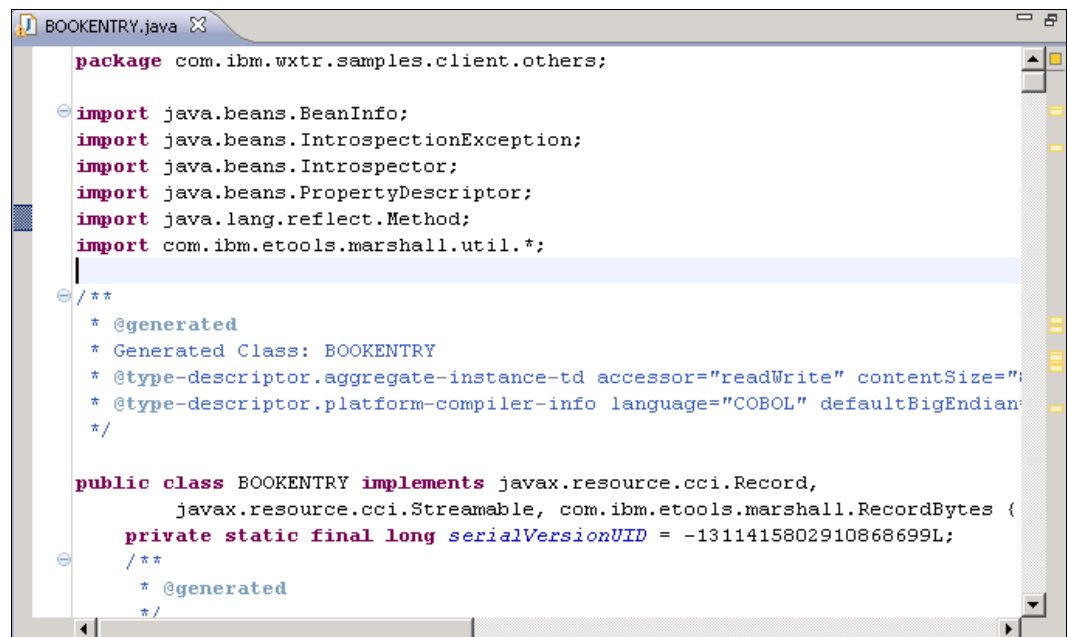


Figure 2-55 Java classes for COBOL copybook specified

For more information and step-by-step instructions about using the J2C Java Data Binding wizard, see the developerWorks® article at this website:

http://www.ibm.com/developerworks/websphere/library/techarticles/0509_barosa/0509_barosa.html

You can also see the Shopping Mall sample that is shipped with WXTR to learn more about the J2C Java data binding tool.

2.4.4 Accessing CICS COBOL through JCA and SCA

CICS COBOL programs that are deployed on WXTR can be accessed by external applications such as web pages and mobile devices through the JCA and SCA interfaces provided by WXTR.

WXTR implements a JCA and an SCA. The WXTR JCA implements a standard resource adapter and the CCI that connects to EISs, such as CICS. The WXTR SCA provides three APIs to access COBOL applications that are deployed on WXTR.

JCA interface

WXTR ships with a JCA Resource Adapter to act as an interface between applications. WXTR also includes an EIS that acts as an integration layer between WXTR and WebSphere Application Server.

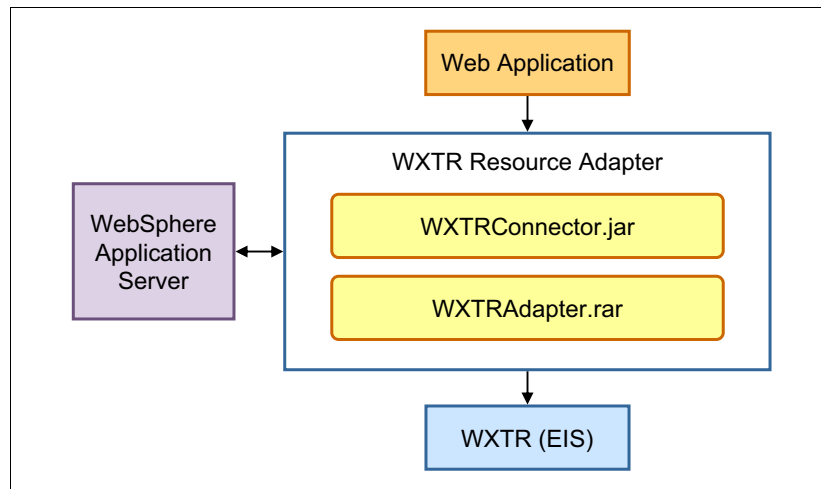


Figure 2-56 WXTR JCA architecture

As shown in Figure 2-56, the JCA Resource Adapter features the following primary components:

- ▶ `WXTRAdapter.rar` is the component that creates and maintains the socket connection between a WebSphere Application Server application and a WXTR application server instance.

WXTRAdapter must be deployed as a resource adapter on WebSphere Application Server. Complete the following steps to deploy WXTRAdapter:

- Connect and log in to the WebSphere Application Server administrative console by using the following URL: `http://server-name:admin-port/admin` (where `server-name` is the hostname or IP address of the machine on which WebSphere Application Server is running and `admin-port` is the administrative console port).
- Select **Resources** → **Resource Adapters**.

- c. Click **Install RAR**.
 - d. If the browser is opened on the AIX server in which WXTR is installed, select **Local file system**. If the browser is opened on a system that is accessing WXTR remotely, select **Remote file system**.
 - e. Select the adapter at <WXTR Install Directory>/lib/WXTRAdapter.rar and click **OK**.
 - f. Click **Next** → **OK** and save the adapter to the master configuration.
 - g. Restart WebSphere Application Server.
- WXTRConnector.jar contains the CCI classes that provide a programmatic interface for accessing the JCA Resource Adapter. Example 2-27 contains a code snippet that describes the use of the WXTR CCI Classes.

Example 2-27 Code to create a JCA connection using WXTRConnector.jar

```

/*
 * ConnectionFactory object of the WXTR JCA adapter.
 */
    WXTRConnectionFactory wxtrf=null;
    WXTRConnection wxtrc =null;
/*
 * Getting the ConnectionFactory object in non-managed connection mode
 */

    WXTRManagedConnectionFactory wxtrmcf = new WXTRManagedConnectionFactory();
    try {
        wxtrf = (WXTRConnectionFactory)wxtrmcf.createConnectionFactory();
    } catch (ResourceException e) {
        e.printStackTrace();
    }

    /*
     * creates a connection object from the ConnectionFactory object
     */
    try {
        wxtrc=(WXTRConnection)wxtrf.getConnection();
    } catch (ResourceException e) {
        e.printStackTrace();
    }

```

The following classes are required to create a connection between the JCA adapter and an enterprise information system:

- The WXTRConnectionFactory object establishes the connection to the EIS (WXTR container). A WXTRConnectionFactory object is used to obtain a Connection object. In Example 2-28 on page 70, the WXTRConnectionFactory is created by using a WXTRManagedConnectionFactory object. WXTR provides a non-managed environment, so the WXTRManagedConnectionFactory object must be used to create the WXTRConnectionFactory object.
- The WXTRConnection object represents the actual connection to WXTR. It provides a method, getConnection, to create the WXTRConnection object, as shown in Example 2-28 on page 70. This object provides more mechanisms to customize the connection to an EIS and create an interaction record.

- The `WXTRInteractionSpec` object is created to set the properties of a request that is sent to WXTR. By using this object, you can set the program name and define whether the request is synchronous or asynchronous. It is passed as a parameter to the `execute` request that is called by a `WXTRInteraction` object. Example 2-28 shows the creation of a `WXTRInteractionSpec` object and setting the program name to be called in WXTR.

Example 2-28 Creating an Interaction request to call a WXTR resource

```

/*
 * InteractionSpec object of the WXTR JCA adapter.
 */
WXTRInteractionSpec wxtrspec = new WXTRInteractionSpec();
wxtrspec.setProgramName("HELLOWLD");
/*
 * Input and Output records to pass/receive data from WXTR
 */
JavaStringRecord input = new JavaStringRecord();
JavaStringRecord output = new JavaStringRecord();
input.setText("Hello World");
try {
/*
 * The object "wxtrc" was created in Example 2-28 on page 70
 */
WXTRInteraction wxtr = (WXTRInteraction)wxtrc.createInteraction();
output = (JavaStringRecord)wxtr.execute(wxtrspec, input);
} catch (ResourceException e) {
System.out.println("The wxtrc.createInteraction() or wxtr.execute()
failed");
e.printStackTrace();
}

```

- The `WXTRInteraction` class is used to create an interaction with the connection already established by using the `WXTRConnection` object. The `execute` method is invoked by using the `WXTRInteraction` object. This process initiates the socket connection to the WXTR container. Depending on whether the request is synchronous or asynchronous (as defined in `WXTRInteractionSpec`), the control returns immediately or after the response is received.

SCA interface

WXTR provides an SCA interface with a `WXTRService` component that enables other SOA components to access COBOL applications that are deployed as SOA services on WXTR. The services can be bound as SCA or as web services.

To write SCA applications by using the WXTR SCA interface, you need to know how to enable SCA on WXTR and use different bindings and services.

Deploying the WXTR SCA interface

To use the WXTR SCA interface, you need to deploy it on WebSphere Application Server. WXTR supplies the SCA interface implementation in `wxtr sca.jar`. Complete the following steps to deploy `wxtr sca.jar`:

1. Complete the following steps to create an asset:
 - a. Connect and log in to the WebSphere Application Server administrative console by using the following URL: `http://server-name:admin-port/admin` (where `server-name` is the hostname or IP address of the workstation where WebSphere Application Server is running and `admin-port` is the administrative console port).
 - b. Select **Applications** → **Application Types** → **Assets**.
 - c. If the browser is opened on the WXTR-installed system, select **Local file system**. If it is opened on a system that is accessing WXTR remotely, select **Remote file system**.
 - d. Choose the `wxtr sca.jar` file in the `<WXTR Install Directory>/lib` directory.
 - e. Click **Next** → **Next** → **Finish**, which accepts the default selections each time. You see a message that states that the asset is now successfully deployed.
 - f. Click **Save** to save the asset to the master configuration.
2. Complete the following steps to create the Business Level Application (BLA) and attach the asset to it:
 - a. On the administrative console, select **Applications** → **Application Types** → **Business-level applications**.
 - b. Click **New**, specify `wxtr sca` in the Name field, and then click **Apply**.
 - c. In the General properties area, under Deployed assets, click **Add** and select **Add asset**.
 - d. Choose `wxtr sca.jar` from the list. Click **Continue** → **Next** → **Next** → **Next** → **Next** → **Finish**, which accepts the default selections each time. You see a message stating that the operation completed successfully.
 - e. Click **Save** to save the BLA into the master configuration.
 - f. Select **wxtr sca BLA application**, click **Start**, and then check to confirm that the application launched successfully.

Using WXTR SCA bindings and services

WXTR enables multiple client types by providing a default SCA binding and a web services binding.

To use the SCA binding in an application, you must create a `WXTRService` object that contains the interfaces to call a WXTR COBOL program as a service. Complete the following steps to create a `WXTRService` object by using Rational Application Developer as the development environment:

1. Include `wxtr sca.jar` in the library path. Click the project in the left panel of Rational Application Developer and select **Project** → **Properties** → **Java Build Path** → **Libraries** → **Add External Jars** and add `wxtr sca.jar`.
2. Select **User Web project** → **Project properties** → **Project Facets**, and then select the following check boxes:
 - Service Component Architecture (SCA)
 - Either of the following information: WebSphere 7.0 Feature Pack for SCA (if you are using version 7.0) or WebSphere 8.0 Beta SCA (if you are using version 8.0).

3. Use the `CurrentCompositeContext` APIs to obtain the `CompositeContext`, which can be used to create the `WXTRService` object. Example 2-29 is a snippet from a servlet that creates a `WXTRServiceComponent` that is bound to the default SCA binding.

Example 2-29 A snippet from a servlet to create WXTRService component

```
import com.ibm.wxtr.sca.WXTRServiceException;
import com.ibm.wxtr.sca.WXTRService;
import com.ibm.websphere.sca.context.CompositeContext;
import com.ibm.websphere.sca.context.CurrentCompositeContext;

/*
** Create WXTRService component SCA service. It must be bound to default SCA **
binding.
*/

CompositeContext compositeContext = CurrentCompositeContext.getContext();
WXTRService wxtrService = (WXTRService)
    compositeContext.getService(WXTRService.class,
        "WXTRServiceComponent/WXTRService");
```

To use web service binding, you need to deploy `wxtrscs`, add a web services client to the existing project, and create the `WXTRService` service. You use one of the following options to deploy `wxtrscs`:

- ▶ Create a web service client in a Rational Application Developer project by completing the following steps:
 - a. In Rational Application Developer, click the project and select **File** → **New** → **Web Service Client**.
 - b. Add the wsdl URL against Service Definition in the following format:
`http://server-name:http-port/WXTRServiceComponent/WXTRService?wsdl`
- ▶ Create a `WXTRService` service with a web services client and declare it as shown in Example 2-30.

Example 2-30 A snippet from a servlet to create WXTRService in a Web Service client

```
import com.ibm.wxtr.WXTRServiceImpl;
import com.ibm.wxtr.WXTRService;
....
WXTRServiceImpl service = new WXTRServiceImpl();
WXTRService wxtr = service.getWXTRServicePort();
```

WXTRService interfaces

`WXTRService` service provides the following interfaces to call applications that feature different types of `COMMAREA` data:

- ▶ `void invoke(String programName)`, which throws a `WXTRServiceException`

This method is used when no `COMMAREA` data is being passed to the application that is deployed on `WXTR` and the application expects no data to be returned from `WXTR`. The following parameters are used:

 - Input parameters: `programName` (the name of the application that is deployed on `WXTR`)
 - Output parameters: None

- `String invokeWithStringData(String programName, String inputRecord)`, which throws a `WXTRServiceException`

This method is used when string-type COMMAREA data is being passed to and returned from WXTR. The following parameters are used:

- Input parameters: `programName` (the name of application that is deployed on WXTR) and `inputRecord` (the string data that is being sent to a WXTR application as COMMAREA).
- Output parameters: A string record is returned as COMMAREA by the WXTR application.

- `byte[] invokeWithByteData(String programName, byte[] inputRecord)`, which throws a `WXTRServiceException`.

This method is used when a byte array is passed to the WXTR application as COMMAREA and the same is expected to be returned. The following parameters are used:

- Input parameters: `programName` (the name of application that is deployed on WXTR) and `inputRecord` (the byte array data that is being sent to a WXTR application as COMMAREA).
- Output parameters: A byte array record is returned as COMMAREA by the WXTR application.

Error propagation

WXTR provides a unique feature for propagating COBOL runtime errors and application-specific errors to a Java EE application for further processing. These error messages use the JCA standard interface, but appear as `javax.resource.ResourceExceptions`, as shown in Figure 2-57.

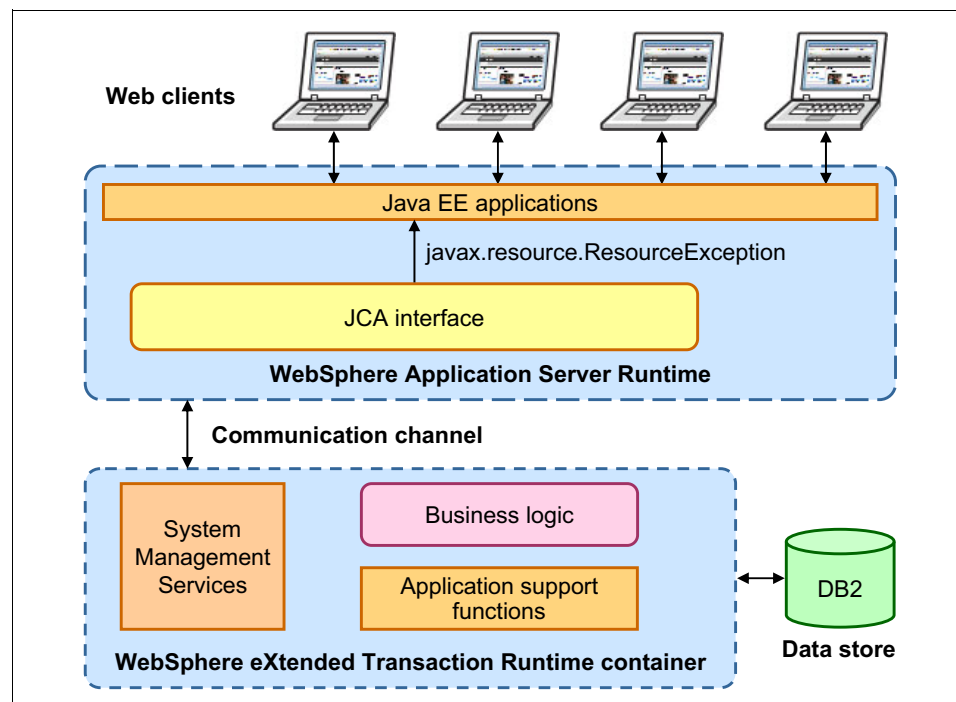


Figure 2-57 CICS COBOL error propagation to Java EE applications by using JCA

For the WXTR SCA interface, the errors are captured as `com.ibm.wxtr.COBOLExecExceptions`. This error propagation process is shown in Figure 2-58.

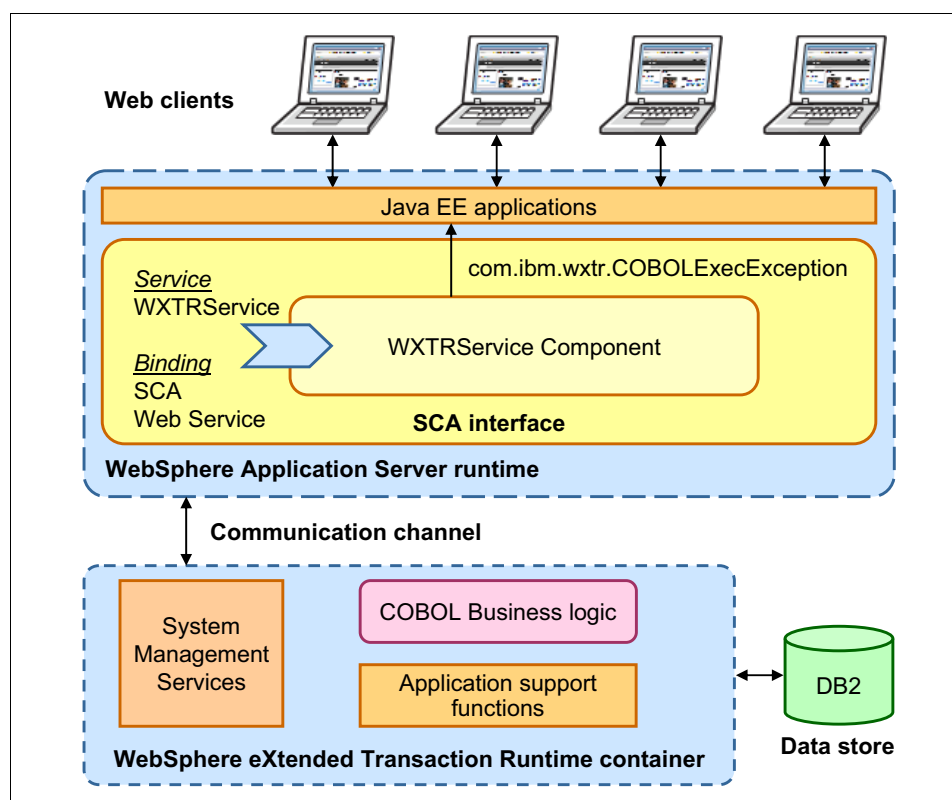


Figure 2-58 CICS COBOL error propagation to Java EE applications by using SCA

The error messages consist of an error code (the abend code that is generated by the CICS COBOL application) followed by text that explains the error. For all COBOL runtime errors, the abend code A012 is generated. Example 2-31 shows a sample error message that is propagated to the Java EE application.

Example 2-31 COBOL application failed with Decimal-divide exception

```

IWZ902S The system detected a Decimal-divide exception.
Message routine called from offset 0x40 of routine iwzWriteERRmsg.
iwzWriteERRmsg called from offset 0x1ee0 of routine _iwzcBCD_DIV_Pckd.
_iwzcBCD_DIV_Pckd called from offset 0x2fc of routine DIVZ.
IWZ901S Program exits due to severe or critical error, error code: A012

```

In Example 2-31, A012 is the abend code and IWZ902S is the COBOL runtime error code.

Application and COBOL runtime error messages are logged in the console file that is in the WXTR container directory. For each restart, the file name gets a new number, such as console.000000.

For more information about error propagation, see the WXTR information center at this website:

http://publib.boulder.ibm.com/infocenter/wxtformp/v1r0/topic/com.ibm.cics.wxt.doc/reference/r_ccwas_err_msg_abnd.dita.html

2.4.5 Creating an enterprise application archive

Enterprise applications are Java EE applications that can be deployed onto an application server. A deployable enterprise application can consist of code artifacts, such as enterprise beans, servlets, JavaServer Pages (JSP) files, and other web components, resource adapter (connector) implementations, application clients, and supporting classes and files. Enterprise applications are sometimes referred to as EAR files because enterprise application projects are stored in enterprise archive (EAR) files.

A Java EE application that is created for WXTR can be deployed in two ways: directly to WebSphere Application Server from Rational Application Developer (by choosing the specific WebSphere Application Server to which WXTR is configured on the AIX machine), or as an EAR file that is deployed to the appropriate WebSphere Application Server that is using the administrative console.

Creating the EAR

Complete the following steps to create an EAR file:

1. Launch Rational Application Developer on your workstation.
2. If you have not yet created an enterprise application project, select **File → New → Project → Java EE → Enterprise Application Project**, and then click **Next**. If you have already created a project, click **File → New → Enterprise Application Project**.
3. In the Create an EAR application section of the New EAR Application Project dialog (as shown in Figure 2-59 on page 76), complete the following steps:
 - a. Specify the EAR project name in the Project name field.
 - b. If you need to change the default project location, click **Browse** and specify a new location.
 - c. Select **WebSphere Application Server v7.0 stub** or **WebSphere Application Server v8.0 stub** in the Target runtime field.

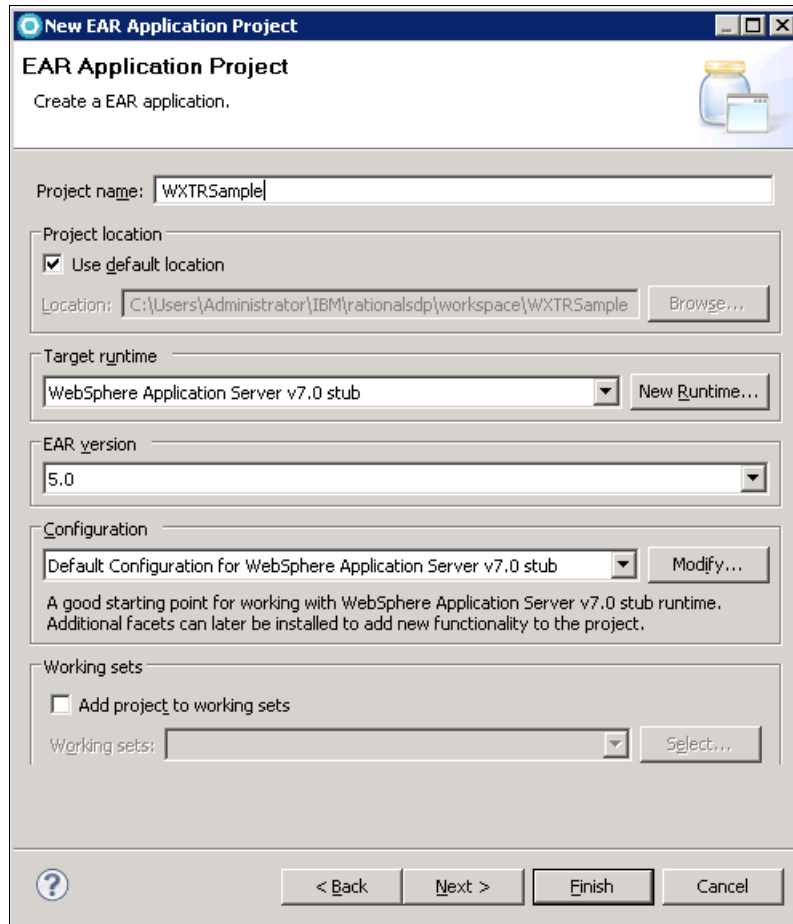


Figure 2-59 Creating an EAR file

4. Click **Next**.
5. In the Configure enterprise application settings section of the window (see Figure 2-60 on page 77), select the modules that you want to add to the project from the Java EE module dependencies list. If you need a different module, click **New Module** and select the module that you need.

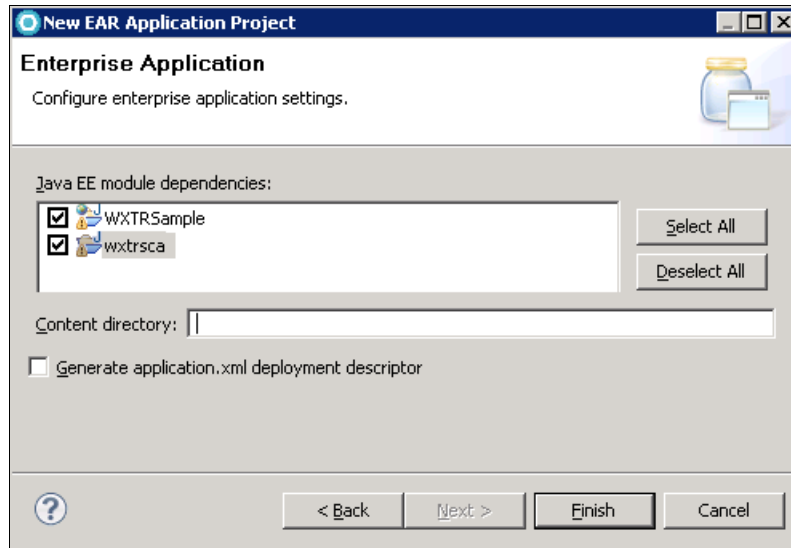


Figure 2-60 Selecting existing modules for creating an EAR

6. Click **Finish**. The EAR project is created and listed in the Enterprise Explorer view.
7. Locate the EAR project in the list, right-click it and then select **Export** → **EAR** file.
8. In the EAR Export window, click **Browse** to select a destination for the EAR project you are exporting. Select the **Export source files** and **Overwrite existing file** check boxes.
9. Click **Finish**. The EAR file is saved in the specified location.

Deploying the EAR

You can use the WebSphere Application Server administrative console to deploy the EAR file to the WebSphere Application Server to which WXTR is configured. Complete the following steps to deploy the EAR file:

1. Connect and log in to the administrative console by using the following URL:
<http://hostname:port/admin> (where hostname is the name or IP address of the machine where WebSphere Application Server is running and port is the administrative console port).
2. On the left side of the window, click **Applications** → **New Application**. On the right side of the window, click **New Enterprise Application**, as shown in Figure 2-61 on page 78.

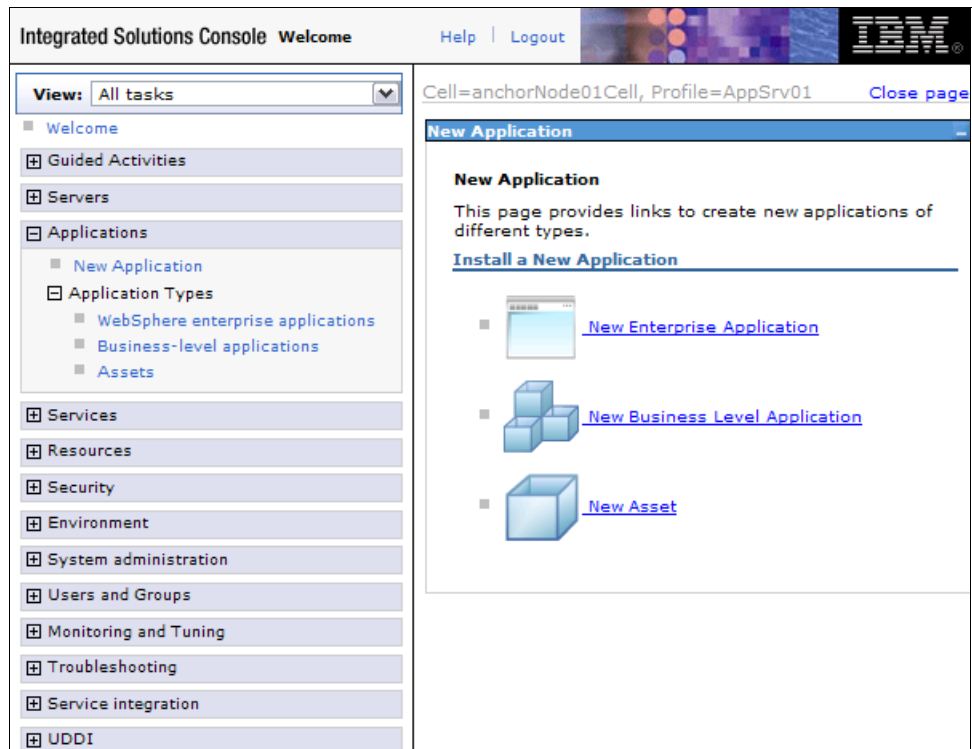


Figure 2-61 Deploying the EAR file

3. Provide the path to the location of the EAR file or click **Browse** to navigate to the path (as shown in Figure 2-62). Click **Next**.

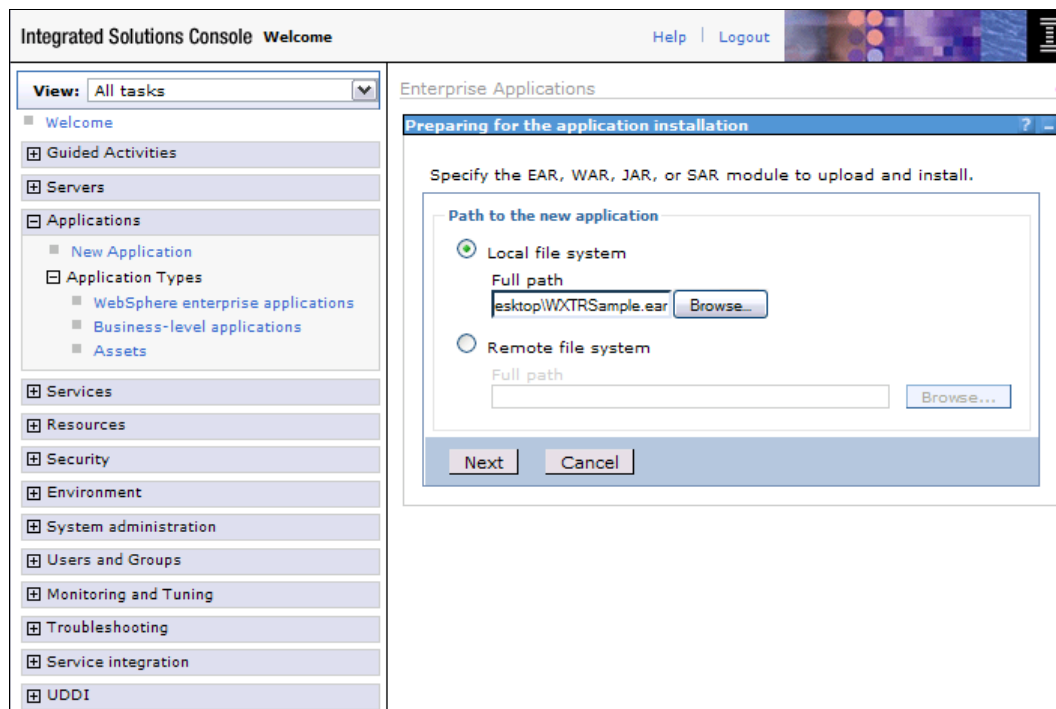


Figure 2-62 Selecting EAR file to deploy on WebSphere Application Server

4. Under **Preparing for the application installation**, select **Fast Path**. Click **Next**.

5. Under **Select installation**, click **Next**, which retains the default selections.
6. Under **Map modules to servers**, click **Next** (retaining the default options).
7. When the summary is presented, click **Finish** and then check to confirm that the application is installed successfully.
8. Click **Save** to save the changes.
9. Return to the administration console and click **Applications** → **Application Types** → **WebSphere enterprise applications**. Select the web application and start it.

Important: If you want to know the context root of the deployed web application, click **Applications** → **Application Types** → **WebSphere enterprise applications** and then select the web application. Select **Configuration** → **Web Module Properties** → **Context Root For Web Modules** and check the context root. If you want to modify the root, change the name and click **OK**. When you are done, check the configuration changes messages and click **Save** to save the changes.



Comparing WXTR with TXSeries for Multiplatforms

In this chapter, we introduce IBM TXSeries for Multiplatforms and compare it with WebSphere eXtended Transaction Runtime (WXTR). Use cases and scenarios are presented for each product to help you identify which product best fits your needs.

The chapter includes the following sections:

- ▶ Introduction to TXSeries for Multiplatforms
- ▶ Comparison of WXTR and TXSeries functionality
- ▶ Considerations when selecting WXTR versus TXSeries
- ▶ Typical configurations for WXTR and TXSeries
- ▶ WXTR constraints
- ▶ Conclusion

3.1 Introduction to TXSeries for Multiplatforms

TXSeries for Multiplatforms is a distributed Online Transaction Processing (OLTP) environment for mixed language applications. The TXSeries architecture of integrated software components can be used to create a Customer Information Control System (CICS) environment. It is widely used for integrating data and applications between distributed solutions and enterprise systems.

TXSeries for Multiplatforms provides the following capabilities:

- ▶ Managed transaction processing environment
- ▶ Composite application-serving infrastructure
- ▶ Deployment platform for C, C++, Java, COBOL, and PL/I applications

TX Series for Multiplatforms maximizes the effectiveness of applications and employees skilled in the AIX, Windows, Solaris, HP/UX PARISC, and HP/UX Itanium platforms. It also allows you to scale up to a centralized CICS Transaction Server for z/OS as your business grows.

A high-level view of TXSeries for Multiplatforms is shown in Figure 3-1.

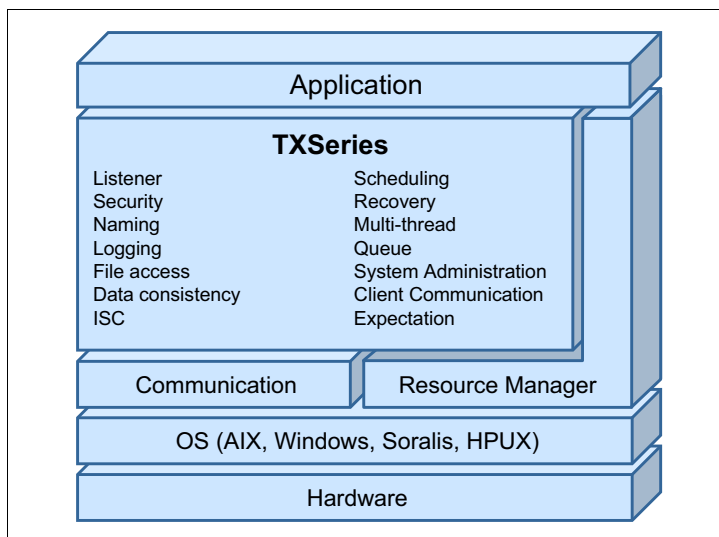


Figure 3-1 High-level view of TXSeries for Multiplatforms

Clients, other TXSeries systems, and CICS Transaction Server (CICS TS) systems can connect to a TXSeries system. TXSeries client connectivity options are listed in Table 3-1.

Table 3-1 TXSeries client connectivity options

Option	Details
Local terminal	A local CICS client 3270 terminal
CICS Transaction Gateway (CTG) and CICS TG Desktop Edition	CICS Family client products that provide a 3270 terminal function and client API
Telnet client	Traditional green screens that are based on 3270 terminal can be used to connect to TXSeries

TXSeries connectivity options are shown in Figure 3-2.

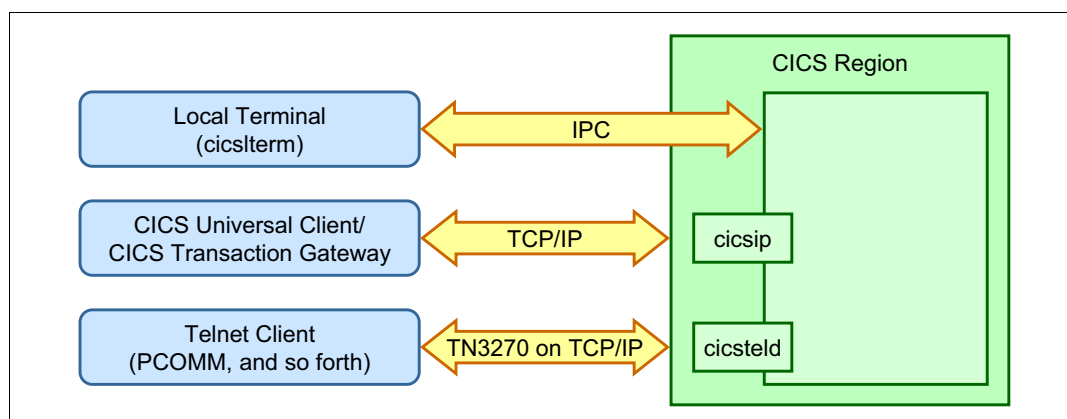


Figure 3-2 Client connectivity on TXSeries for Multiplatforms

3.2 Comparison of WXTR and TXSeries functionality

WXTR and TXSeries for Multiplatforms are transaction processing products, but there are important differences between them. This chapter highlights the differences between these products and CICS Transaction Server (CICS TS).

3.2.1 CICS API support

This section compares CICS API support for each of the products. TXSeries supports a subset of the CICS APIs that CICS TS supports for compatibility purposes. WXTR supports this same subset of APIs except Basic Mapping Support (BMS), Intersystem communication (ISC), terminal, and security, as shown in Figure 3-3.

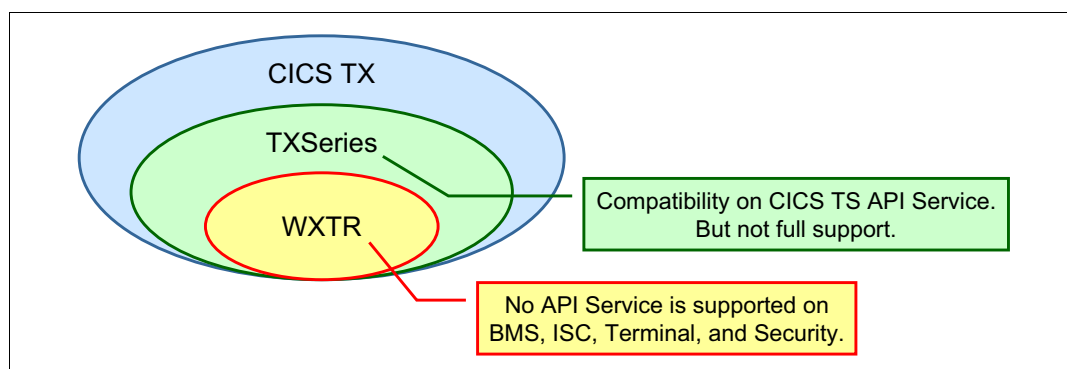


Figure 3-3 CICS API support

3.2.2 Migration from CICS servers

TXSeries is the ideal choice if you still need to use the 3270 terminal, BMS, or Intersystem ISC function. Otherwise, WXTR is more suited for CICS application modernization on distributed platforms for migrated applications.

3.2.3 Integration with WebSphere Application Server

WXTR is tightly integrated with WebSphere Application Server. TXSeries is combined with CICS Transaction Gateway (CTG) for transaction processing. A typical WebSphere Application Server with TXSeries configuration is shown in Figure 3-4.

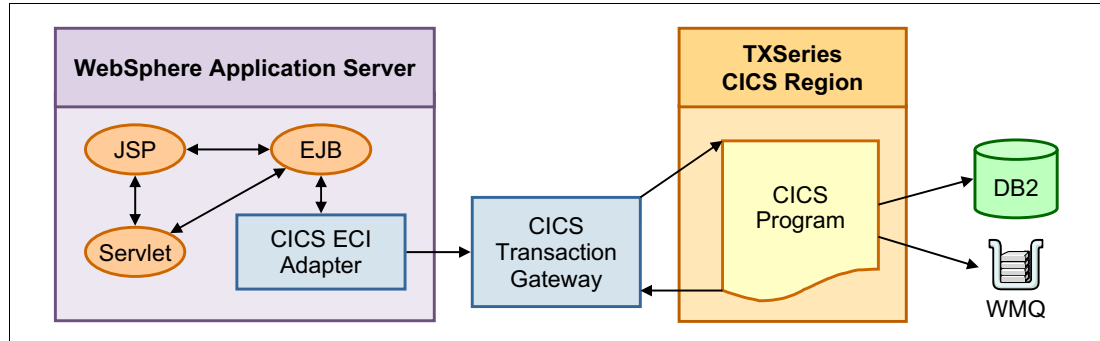


Figure 3-4 WebSphere Application Server with TXSeries configuration

WXTR must be configured on the same node as WebSphere Application Server. However, there is no similar requirement for TXSeries and CTG to be configured on the same server with WebSphere Application Server or each other.

TXSeries provides configuration flexibility. WXTR offers better performance and configuration simplicity.

3.2.4 Supported software

The software that is supported by each product is compared in Table 3-2. TXSeries requires a CICS client product, CTG, to communicate with WebSphere Application Server.

Table 3-2 Supported software on WXTR and TXSeries

Software	WXTR	TXSeries
Operating system	AIX	AIX, Windows, HP/UX, Solaris
Compilers	IBM COBOL, C	IBM COBOL, C, C++, MicroFocus Express COBOL, IBM PL/I, Java
Database	IBM DB2, Oracle	IBM DB2, Oracle, Informix® Client SDK, Sybase
Application server	IBM WebSphere Application Server	Not Applicable (depends on CTG supported software)
Development tool	IBM Rational Developer for Power System (RDp)	IBM Rational Developer for Power System (RDp)
Integration middleware	Not Applicable	WebSphere MQ
Install requirements	IBM Installation Manager	Not Applicable

The following listing of supported software is current as of the time of this writing:

- ▶ WXTR

<http://www-01.ibm.com/software/webservers/appserv/extended-transaction-runtime/requirements.html>

- ▶ TXSeries for Multiplatform

<http://www-01.ibm.com/software/http/cics/txseries/requirements.html>

- ▶ CICS Transaction Gateway

<http://www-01.ibm.com/software/http/cics/ctg/reqs/>

3.2.5 System management functionality

WXTR supports a subset of the CICS system management APIs that TXSeries supports. Table 3-3 shows the comparison of CICS functionality that is focused on the system management features of WXTR (integrated with WebSphere Application Server) and TXSeries (configured separately).

Table 3-3 System management comparison

Feature	WXTR	TXSeries
Start / Stop	Start and stops automatically as managed in the WebSphere Application Server. Commands: <ul style="list-style-type: none">▶ startCICSServices▶ stopCICSServices	Start and stops independently. Commands: <ul style="list-style-type: none">▶ cicscp▶ SMIT▶ startsrc
Defining resources	WebSphere Application Server administrative console Command: wsadmin command	Web Administration Console Commands: <ul style="list-style-type: none">▶ SMIT▶ cics command
Trace	System trace	System Trace, Application Trace
Dump	Supported	Supported

3.2.6 Tooling for developers

WXTR does not support the BMS/3270 terminal because WXTR's CICS API coverage is focused on modernization. TXSeries features a broader range of CICS API support that is designed for reutilization, such as migrating from CICS Transaction Server application to TXSeries.

From a tooling perspective, IBM Rational Developer for Power Systems (RDp), including Rational Application Developer, can be used for WXTR and TXSeries application development. The use of RDp to develop WXTR CICS applications includes the following advantages:

- ▶ Mixed language debugging

A developer can start and inspect the COBOL and Java EE programs by using the same integrated development environment.

Important: One program can be debugged at a time with WXTR by using IBM Rational Developer for System Power (RDp). For more information about debugging, see 2.4, “Modernizing applications” on page 36.

- ▶ Makefile template
RDp provides the template of the makefile for the Make utility, which is used to build the corresponding applications.
- ▶ CICS syntax checker
RDp scans CICS commands for syntax errors.
- ▶ Automatic deployment
After the application is built, deployment occurs automatically when you save or build the project.

3.3 Considerations when selecting WXTR versus TXSeries

This section describes some of the typical considerations when you are selecting WXTR or TXSeries for Multiplatforms. All deployment scenarios are not covered. Instead, the focus is on the most common scenarios.

3.3.1 Advantages of each approach

For the following considerations, WXTR is a better choice than TXSeries for Multiplatforms:

- ▶ You want to modernize existing CICS COBOL applications.
WXTR provides a robust, transactional COBOL container that runs as an extension to WebSphere Application Server. It provides interfaces to expose the COBOL applications as components by using its Service Component Architecture (SCA) interfaces. This design means that the existing COBOL applications can easily be made part of a Service Oriented Architecture (SOA) solution. This ability helps in the process to modernize existing and newly written COBOL business assets.
- ▶ You want to simplify product integration.
WXTR is tightly integrated with WebSphere Application Server and features a common administrative console that is used to configure both run times. Although TXSeries for Multiplatforms can be integrated with WebSphere Application Server, it requires an additional product, such as a CICS Transaction Gateway (CTG) to act as an integrating agent between them. This necessity results in a loosely coupled integration in which each product must be maintained and configured separately, which, in turn, increases complexity and results in longer turn-around times for service and problem determination.
- ▶ You want to improve tooling support for application development.
WXTR supports application development through RDp, so the development and debugging of Java Platform, Enterprise Edition applications and COBOL applications can be done by using a single tool.
- ▶ You need to reuse existing CICS COBOL assets and move them to a distributed platform.
- ▶ The presentation logic is already provided by a Java application (WebSphere Application Server).
- ▶ The future vision is to access CICS COBOL assets as a web service.

- ▶ Your goals include simpler administration.

For the following considerations, TXSeries has advantages over WXTR:

- ▶ You need to use TXSeries as an intermediate server and use intersystem communication (ISC) to communicate with CICS TS.

TXSeries must be used if ISC is needed to communicate with CICS TS or another TXSeries region.

- ▶ You need to reuse a COBOL application.

TXSeries supports more CICS APIs than WXTR, so it is a better choice when a CICS TS application is reutilized.

Important: TXSeries for Multiplatforms supports only a subset of CICS TS APIs.

- ▶ The COBOL application accesses multiple resources, such as WebSphere MQ or a DB2 database.
- ▶ The customer requires a proven transaction processing monitor on a distributed platform.

Important: Transaction processing monitor (TP Monitor) is the product that includes the Transaction Processing function, such as CICS and IMS.

3.3.2 Key points to decide

Although the scenarios that were presented in 3.3.1, “Advantages of each approach” on page 86 are likely situations for using each product, customer configurations are unique, so there cannot be a single set of guidelines. Instead, ask yourself the following questions to help identify the best product for your situation:

- ▶ What do you want to do with your system? Modernize? Replace? Rehost?
- ▶ What is your current environment?
 - Does your environment use a CICS COBOL or C application? Another COBOL or C application?
 - Which application model are you using?
 - How good are your development tools?
 - What are your administrative capabilities?
- ▶ What combination of software support is needed?
 - What operating systems are required?
 - Which resource manager do you want to use? Which compiler?

3.4 Typical configurations for WXTR and TXSeries

This section describes the typical configurations of WXTR and TXSeries.

3.4.1 WXTR configurations

This section describes the basic configuration for WXTR and a custom configuration.

Basic configuration

An instance of WXTR is configured to a specific instance of WebSphere Application Server.

Figure 3-5 shows the basic configuration of WXTR. A WXTR application can be called from a Java EE application via either of the following interfaces:

- ▶ Java EE Connector Architecture (JCA)
- ▶ Service Component Architecture (SCA)

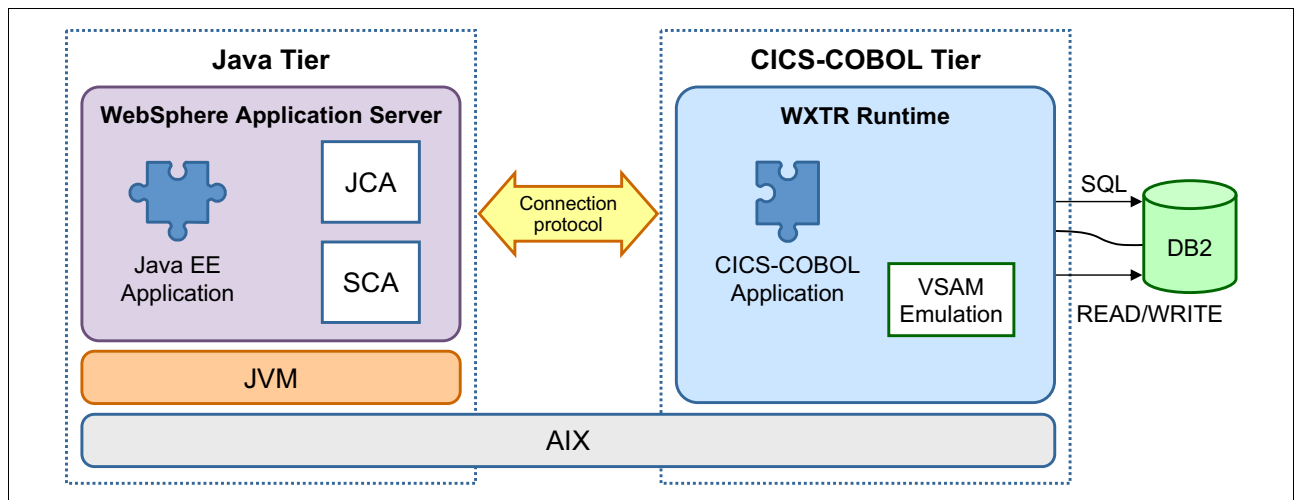


Figure 3-5 A basic configuration of WXTR for use with a COBOL application

A COBOL application that is running in WXTR can access data that is in a DB2 database. VSAM file access is achieved through the emulation layer that is provided for DB2.

VSAM file: VSAM file is the file that is managed by the same file organization that is used on mainframe.

Custom configuration

For every WXTR installation, there must be a corresponding WebSphere Application Server installation. Figure 3-6 shows a configuration that is designed for high availability. In this topology, a load balancer distributes work among multiple web servers. Each web server sends the work to a specific application server. Fault detection, or the weight that is allocated to each request, depends on the high-availability design.

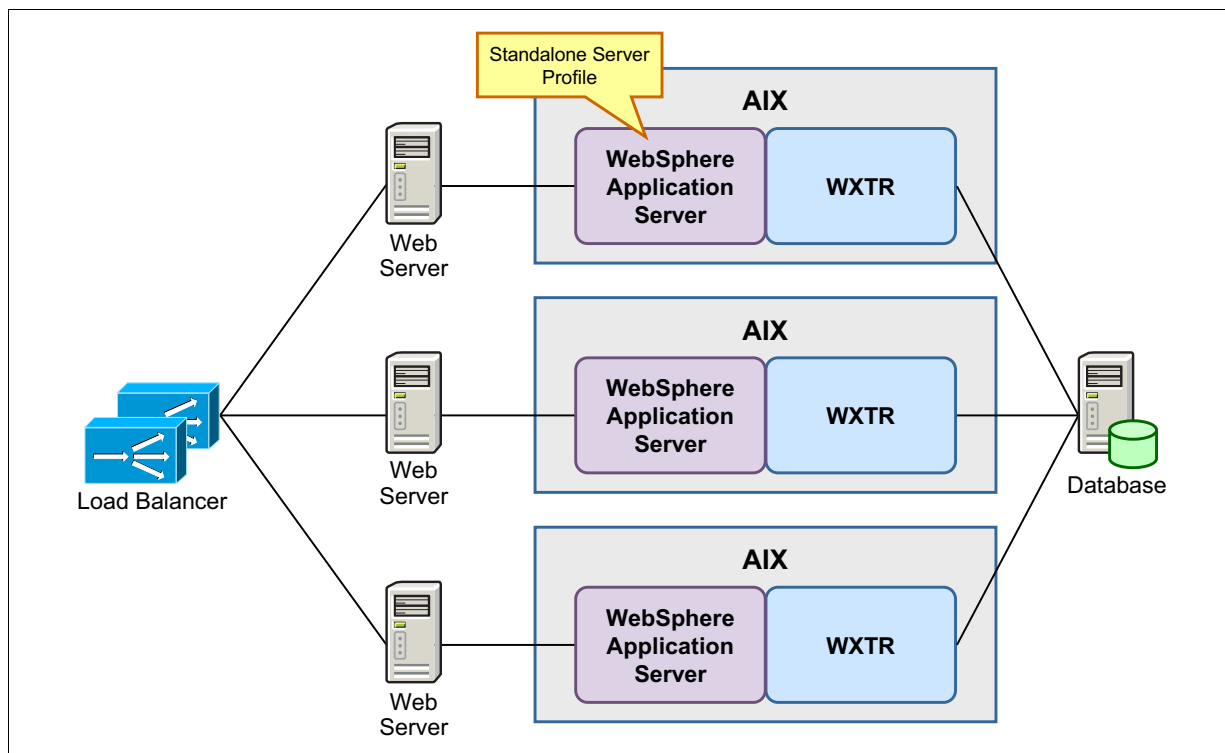


Figure 3-6 Custom WXTR configuration that is designed for high availability

3.4.2 TXSeries for Multiplatforms configurations

The configuration options for TXSeries are described in this section.

Basic configuration with WebSphere Application Server

TXSeries for Multiplatforms includes the following common topologies:

- ▶ Back-end server
- ▶ Middle tier server (in a multitier configuration)

TXSeries for Multiplatforms requires an additional product, a CICS Transaction Gateway (CTG), between it and WebSphere Application Server.

Figure 3-7 on page 90 shows the basic configuration for TXSeries to communicate with WebSphere Application Server. A Java EE application can call a TXSeries application by using the JCA adapter that is provided as part of the CICS Transaction Gateway.

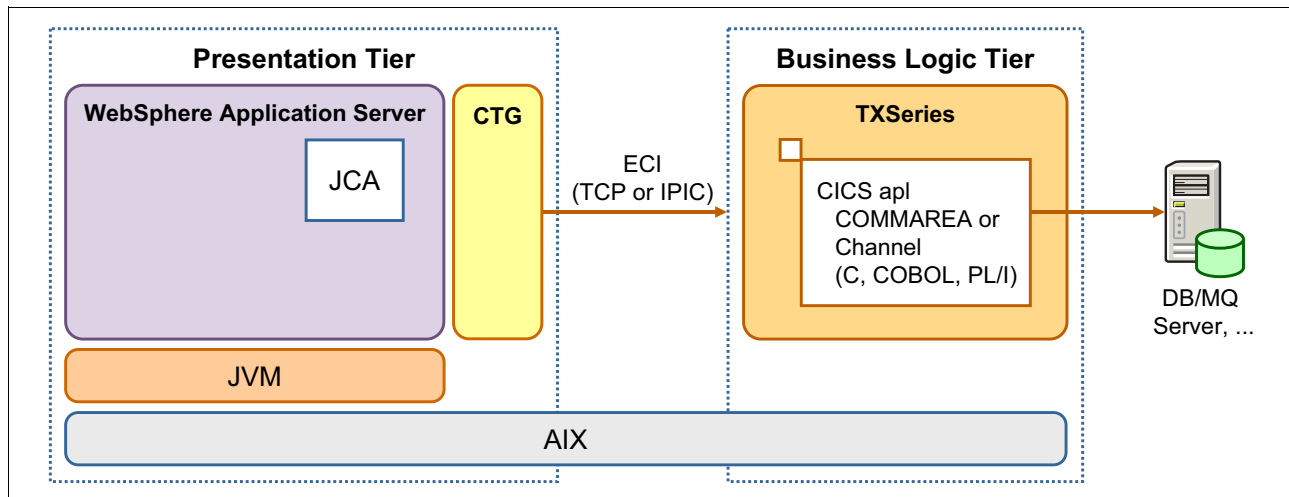


Figure 3-7 Basic configuration on TXSeries

Important: For more information about the CICS Transaction Gateway, see this website:

http://www-01.ibm.com/software/sw-library/en_US/products/G158593Z87967N09/

The primary interface that is provided by the CICS Transaction Gateway is the JCA adapter. For more information about developing applications by using connectors, see *Developing Connector Applications for CICS*, SG24-7714.

Custom configuration

TXSeries for Multiplatforms can work with multiple CICS regions on a single machine. They also can be on a different system from WebSphere Application Server. So, TXSeries for Multiplatforms can scale vertically and horizontally to provide high availability.

Figure 3-8 shows a sample configuration of a high-availability TXSeries system.

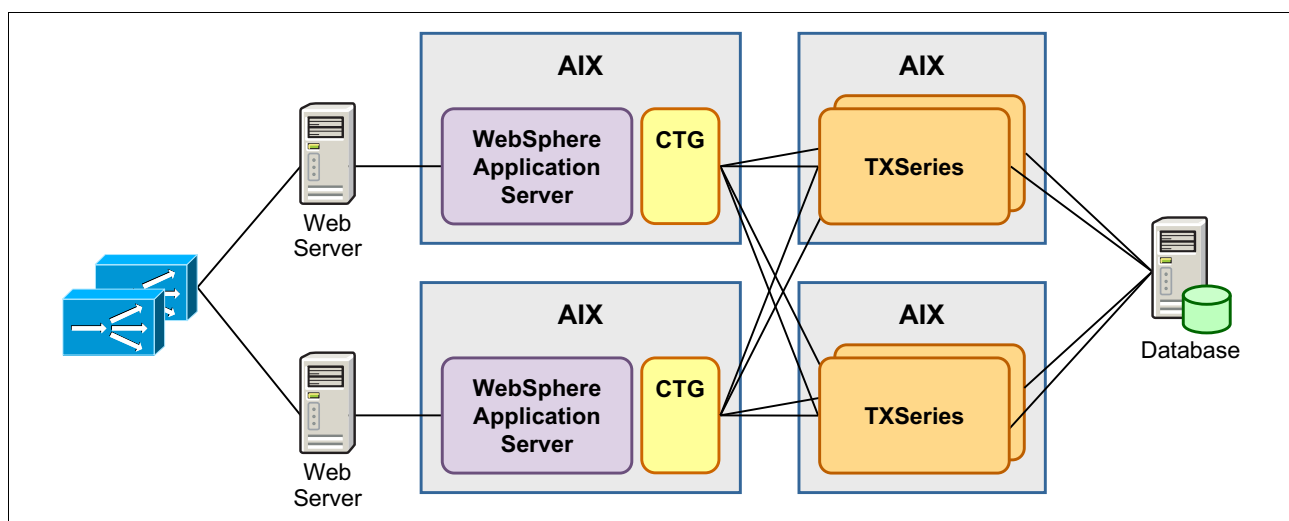


Figure 3-8 Custom TXSeries configuration designed for high availability

Important: Scaling vertically and horizontally demands more system resources (memory, CPU, and so on). Scaling also makes problem determination more complicated than it is under a WebSphere Application Server-WXTR configuration.

3.5 WXTR constraints

This section summarizes the restrictions on the use of WXTR V2.1. For more information, see this website:

<http://pic.dhe.ibm.com/infocenter/wxtrform/v2r1/index.jsp>

3.5.1 Configuration

WXTR configuration constraints include the following aspects:

- ▶ WXTR can be configured on multiple WebSphere Application Server workstations for a profile that is augmented with WXTR.
- ▶ For every WebSphere Application Server instance, there can be only one WXTR instance.

3.5.2 Transactions

WXTR transaction constraints include the following aspects:

- ▶ JCA applications must open a new JCA connection for every program request. Existing connections cannot be reused.
- ▶ Oracle Database or IBM DB2 must be used as the Resource Manager for data access.
- ▶ The WXTR service must be configured with a single database resource manager in XA Interface.

3.5.3 Application support

WXTR application constraints include the following aspects:

- ▶ WXTR supports only 32-bit COBOL and C applications.
- ▶ Not all CICS API functions are available with WXTR.

Important: A list of CICS APIs that are supported in WXTR V2.1 is available in the *Application programming interfaces* topic in the WebSphere eXtended Transaction Runtime Information Center at this website:


<http://publib.boulder.ibm.com/infocenter/wxtform/v1r0/index.jsp>

3.6 Conclusion

TXSeries for Multiplatforms is a distributed CICS OLTP environment for mixed language applications, including COBOL applications. In contrast, WXTR is a distributed OLTP environment that is used to develop and host modern COBOL and C applications.

WXTR delivers a modern, tightly integrated, managed environment for hosting COBOL and C business applications within WebSphere Application Server. This ability enables you to extend your essential applications by using Java EE.

Although WXTR and TXSeries for Multiplatforms include their own features and relative advantages, WXTR is the ideal choice when it comes to modernizing existing C and COBOL applications.



Migrating Java applications from Oracle WebLogic

Existing Tuxedo applications can be moved to WebSphere eXtended Transaction Runtime (WXTR). If the Tuxedo client applications and other associated Java applications are running on any other application server other than WebSphere Application Server, the migration process can be considered complete. However, the migration is complete only when the Java EE applications are also moved to WebSphere Application Server.

This chapter describes the basic concepts of migrating Java applications from WebLogic Application Server to WebSphere Application Server. It gives an overview of the migration process and provides techniques to move WebLogic applications and server configurations to WebSphere Application Server.

This chapter does not offer all possible solutions, nor does it cover every issue that can arise in a migration. It is intended only as a starting point. Information that you need to build a project plan for the most common migration-related activities is provided, and the process of migrating applications to WebSphere Application Server Version 8 is explained. The use of the migration toolkit for Rational Application Developer also is explained.

This chapter contains the following sections:

- ▶ Migration strategy and planning
- ▶ Migration project plan
- ▶ Java EE application migration
- ▶ Post deployment
- ▶ WebSphere Application Server Migration Toolkit
- ▶ Oracle Tuxedo Java client applications
- ▶ Migration of WTC applications to WXTR
- ▶ Conclusion

4.1 Migration strategy and planning

Planning a migration starts with laying out the first tasks that are needed to accomplish it, and then adding other tasks to that list. This process is not intended to be a final methodology, but it helps you to organize the work.

4.1.1 Migration strategy considerations

Many potential problems can be avoided by making informed decisions. Before you begin, it is important to understand the challenges you might face in a migration. Keep in mind the following considerations:

- ▶ Get the applications migrated as quickly as possible and with the fewest changes possible.
- ▶ Do not skip the pre-migration assessment, even if the migration is not considered complex.
- ▶ In the assessment phase, review the existing code to evaluate how well the applications follow, or deviate from, Java EE specifications.
- ▶ Do not forget environments other than production, such as development, QA, and testing. Even if the application code and the application servers (old and new) comply with Java EE specifications, some vendors implement the specifications in their own way or add extensions that also must be migrated.
- ▶ Do not change too much at one time. Use an iterative approach. Migrate a small portion of code and test it before more code is migrated. It is easier to find problems in a small, controlled environment.
- ▶ Keep change variables to a minimum. For instance, during a migration, it is a good idea to freeze non-migration-related code changes. Make sure to evaluate the changes that are needed for build scripts and required runtime configurations.
- ▶ Determine how to deal with Java SE and Java EE specifications. Decide whether you will compile by using compatibility with an earlier version or upgrade instead.
- ▶ Do not replicate your performance tuning from a previous application server. After the migration, make performance tuning decisions that are specific to WebSphere Application Server.

4.1.2 Migration planning

The next step is to plan the migration itself. The planning process and its component parts are shown in Figure 4-1 on page 95.

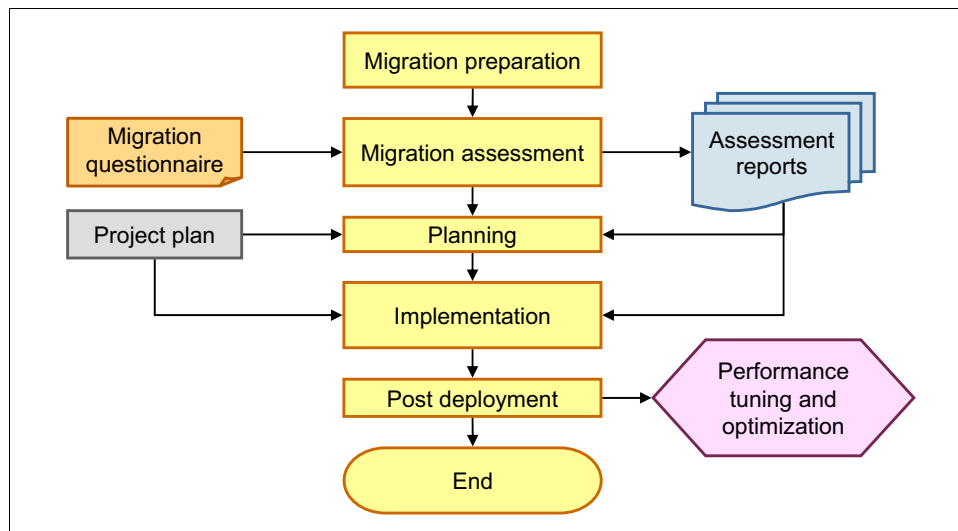


Figure 4-1 High-level migration plan

The following aspects of the initial planning process occur before formal migration planning actually begins:

► Migration assessment

The migration assessment is the starting point of a migration and is intended to help you identify the scope of the project, the expectations for it, the current IT environment, the teams that are involved in the work, and all other relevant information. The aim is to collect whatever information might be useful in planning the migration.

► Migration questionnaire

The migration questionnaire is a set of questions that are designed to gather the necessary information about the migration, business requirements and decisions, hardware and software, topology, architecture, application code, and so on. The following questions are typical for a questionnaire:

- What topology are we planning to use?
- What other environments are involved?
- What are the recommendations for application code migration?
- What configurations need to be done in the runtime migration process?
- Will we create the deployment and administration scripts?

► Migration assessment report

The migration assessment report compiles the relevant answers from the migration questionnaire and other sources, with the aim of identifying and minimizing any threats to the migration effort. The report often includes sections such as Introduction, Education, Sizing, Topology, Application Migration, Runtime (Server) Migration, Testing, Deployment, Tuning and Performance, Risks, and anything else that might be critical to the project.

4.2 Migration project plan

The assessment report helps to identify which activities are dependent upon each other, which activities can run in parallel, and the complexity of each piece of work. All of this information, and more, is included in the migration project plan, which also records decisions that are taken thus far and any decisions that still need to be made.

A good migration project plan should contain sections similar to the following sections:

- ▶ Tasks
- ▶ Test plans
- ▶ Dependencies
- ▶ Risks
- ▶ Production rollouts
- ▶ Checkpoints

4.2.1 Migration implementation

Several factors influence the decision to migrate an application, ranging from business decisions about costs and benefits to the need to adapt to changing IT environments.

Migrating Java EE applications from one application server to another can be simple or complex, depending on the tasks you undertake. But whatever the extent of the migration, it should be done methodically. With appropriate planning, and by adopting an implementation strategy, most negative migration issues can be prevented.

Migration planning and implementation are closely related. The following approach helps to address issues that are discovered in the project plan, and set expectations for future rollouts:

- ▶ Identify possible changes in the environment.
- ▶ Study the runtime and integration questionnaires to see how those answers can help you make the implementation a success; for instance, review the topology and network items.
- ▶ Identify firewall rules that might prevent your application from running appropriately.
- ▶ Determine whether interoperability of servers is required or if integration with existing systems is being addressed.
- ▶ Make sure test plans cover functional and non-functional requirements.
- ▶ Try specific scenarios and separate sources of review to avoid issues when you are moving your application to production environments.

Implementation phases

The iterative process is a good option for migration projects. It helps reduce risk by identifying the critical paths between phases.

In this section, the following typical phases are described:

- ▶ Preparation

In this phase, you check for product and application prerequisites, confirm that you have the necessary hardware, software, and people available, and set up the development environments. You also schedule any required meetings and other administrative tasks.

► Migration

In this phase, the code is moved to the new environment and applications are run in the new environment. The migration and test phases are closely integrated. It can be useful to test small pieces of functionality until you have everything migrated. This approach helps in the following ways:

- Challenges that are encountered in the early stages serve as input to later stages, thus helping to speed up the process.
- Functional tests can be run in code that is migrated in parts. These tests help prepare for QA and load tests after the entire code migration is complete.

► Testing

The testing phase is dynamic and related to the migration and rollout phases. The following main types of tests are in a typical migration:

- Hands on testing, which is done together with functional-regression testing
- Functional-regression testing to make sure that all functions are working as expected
- Load tests to assess whether the application and server are performing as well or better than in the previous environment.

► Rollout

In this phase, the migrated applications are distributed to all of the associated systems, which are part of the entire application infrastructure. This phase starts after all tests are completed and you confirmed that the application performs at the same level, or a better level, than in the previous environment.

Rollouts that involve interoperability and interfaces are more difficult to implement because of potential compatibility problems with existing applications. Be mindful of these challenges and prepare plans to prevent these issues. A good approach is to run the migration rollout during a maintenance window. With this approach, you can stop external inputs to the systems affected, which allows you to test integration and interoperability without having test results that are mixed with valid requests. A cleanup of databases or message queues might also be needed.

► Cleanup

This phase begins after the migrated application is successfully put into production.

Tools that help to monitor performance and system stability can be useful during this phase. Assign specialists to tune application settings as required. Make any required changes in non-production environments first and test the changes there before you extend the changes to the production environment.

4.3 Java EE application migration

Migration of Java EE applications varies in complexity and depends on the following factors, many of which are identified during the migration assessment phase:

- Compatibility of source and target Java EE specifications
- Use of specific Java components (EJB, JSPs, taglibs)
- Use of proprietary code specific to an application server
- Use of vendor-specific deployment descriptors
- Use of extended capabilities
- Use of vendor-specific server configurations (JMS providers, JDBC connection pooling)
- Class loading issues
- Integration with complex external systems such as a CRM or SAP

Performing the necessary changes to address the discrepancies that arise as a result of the points that were mentioned previously, can be done manually or by using tools to speed migration.

IBM Rational Application Developer Version 8.0 features extended functions to help you. A new migration toolkit also was released that helps you migrate applications from Oracle WebLogic and JBoss application servers to WebSphere Application Server V8. For applications that use new features like annotations, Rational Application Developer can provide tags to build and maintain applications that use EJB 3, web services, and other components. Other tools are available to provide guidance in code reviews.

When migrating Java EE application servers, be aware that although the application servers might comply with specifications, each vendor has its own implementation process. Because of this difference, there often are things to change, implement, or migrate. Considering that the application server is a product that runs on an operating system that can also be part of the migration, the changes, and configurations to be made at the OS and application server levels are important.

4.4 Post deployment

Application performance in the new environment should at best match that in the former environment, and preferably it should be better. To achieve this improved performance, you must take more steps with the application and the run time.

Performance tuning and optimization are tasks that require detailed analysis of the environment. A considerable amount of data must be collected to support decisions. This information has two sources: application code review and load test results. With these results, the experts have enough information to make the necessary adjustments.

Again, an iterative process helps. Take small pieces of code or specific configurations and focus on those aspects until you reach the wanted goal. Focus on the critical paths that can lead to better performance for the entire application or environment.

There are tools that help you to achieve performance goals. For more information, see the WebSphere Application Server Information Center at this website:

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.webSphere.nd.multipatform.doc/info/ae/ae/welc_howdoi_tprf.html

4.5 WebSphere Application Server Migration Toolkit

The migration of Java applications to WebSphere Application Server does not have to be an entirely manual process. The IBM WebSphere Application Server Migration Toolkit can help. This suite of tools and helpful information enables organizations to quickly and cost-effectively migrate to WebSphere Application Server V7 or V8, whether from a previous version of the product or from competitive application servers such as Oracle WebLogic Server, Oracle Application Server, and JBoss Application Server.

The migration tools in the kit are built on IBM Rational Software Analyzer, a framework that provides a single solution for identifying, analyzing, and fixing application code quality and compliance requirements. The tools use the scanning capabilities of Rational Software Analyzer to locate application elements that should be updated for optimal compatibility and performance with WebSphere Application Server. They also use the unique editing features of the analyzer to review and appropriately change the code.

The tools are packaged as a feature (a collection of plug-ins) that can be installed into an integrated development environment (IDE) such as Eclipse, Rational Application Developer, or Rational Software Architect. The latest version of the toolkit can be downloaded at no charge from this website:

<http://www.ibm.com/developerworks/websphere/downloads/migtoolkit/index.html>

For more information about installing, configuring, and using the toolkit, see *Application Migration Tools for Competitive Migration*, which is available at this website:

http://public.dhe.ibm.com/software/dw/wes/migrationtoolkit/ApplicationMigrationTool_en_US.3.0.0.pdf

Important: The toolkit is supported by IBM through your existing support entitlement or through a forum that is available at no charge at the following website:

<http://www.ibm.com/developerworks/forums/forum.jspa?forumID=2106>

Another helpful reference is *WebSphere Application Server V8.5 Migration Guide*, SG24-8048, which is available at this website:

<http://www.redbooks.ibm.com/redpieces/abstracts/sg248048.html?open>

The publication includes a chapter titled *Migrating from Oracle WebLogic* that explains the migration process with the help of a sample Java EE application. It refers to IBM WebSphere Application Server Migration Toolkit V1.1, but the process and principles can be easily adapted to later versions.

4.6 Oracle Tuxedo Java client applications

The most commonly used mechanism to call a Tuxedo service from a Java client application is the WebLogic Tuxedo Connector (WTC), which is a component of WebLogic Server. Other commonly used techniques are using web services and Oracle Services Architecture Leveraging Tuxedo (SALT), which allow bidirectional Simple Object Access Protocol (SOAP) connectivity, or by using Oracle Tuxedo Jolt, which provides Java client access to Tuxedo services.

Thus far we saw how to move the Tuxedo server applications to WXTR. In this section, the process that is used to migrate Tuxedo client applications that are written in Java is described.

Oracle Tuxedo provides the following sets of APIs for Java programmers:

- ▶ WebLogic Tuxedo Connector (WTC)
- ▶ Jolt Connector
- ▶ Tuxedo JCA Adapter

As of this writing, WXTR does not provide a utility to assist with the migration of applications that are based on these APIs. For more information and guidelines on migrating a typical WTC-based application, see 4.7, “Migration of WTC applications to WXTR” on page 100.

4.7 Migration of WTC applications to WXTR

The most common mechanism for calling a Tuxedo service from a Java client application is the WebLogic Tuxedo Connector (WTC), which is a component of WebLogic Server. In this section, we will examine a sample WXTR client application. This review provides a base of understanding for migrating WTC application APIs to WXTR. WXTR uses APIs that are defined by the standard JCA specifications. When the sample Oracle Tuxedo code is migrated to WXTR, the code looks like the code shown in Example 4-1.

Example 4-1 Sample WXTR client application

```
private String callWXTRService(String svcName, String input) throws Exception {
    TuxedoConnectionFactory ccfc = null;
    TuxedoConnection ccc = null;
    JavaStringRecord jsr=null;
    String resultString = null;
    1 TuxedoManagedConnectionFactory ccmcf = new
TuxedoManagedConnectionFactory();
    try {
        2 ccfc = (TuxedoConnectionFactory)ccmcf.createConnectionFactory();
    }
    catch (ResourceException e) {
        e.printStackTrace();
        throw e;
    }
    try {
        3 ccc=(TuxedoConnection)ccfc.getConnection();
        4 jsr = new JavaStringRecord();
        jsr.setText(input);
    }
    catch (ResourceException e) {
        e.printStackTrace();
    }
    5 TuxedoInteractionSpec ccispec = new TuxedoInteractionSpec();
    ccispec.setProgramName(svcName);
    try {
        TuxedoInteraction cci = (TuxedoInteraction)ccc.createInteraction();
        6 cci.execute(ccispec, jsr);
    }catch (ResourceException e) {
        e.printStackTrace();
    }
    7 resultString = jsr.getText();
    8 ccc.close();
    return resultString;
}
```

The lines of code are described in the following analyses of the sample WXTR client program that is shown in Example 4-1 on page 100:

- ▶ Lines 1, 2 and 3 gets the connection from the Tuxedo connection factory. WXTR supports the non-managed mode of connection. Unlike the WTC programs, we do not perform a JNDI lookup to get the connection factory because of this configuration.
- ▶ Line 4 uses a Record object, as required by JCA standards, for sending and receiving messages or data between the client application and WXTR. In this example, a `JavaStringRecord` is used.
- ▶ Line 5 creates the interaction specifications object. The name of the program to be invoked is set as part of this interaction-specification creation.
- ▶ Line 6 invokes the program.
- ▶ Line 7 gets the result of the program invocation as a Java string. The client program can use this string for any other processing at the client side.
- ▶ Line 8 closes the server connection and detaches from the current connection object.

In comparison with a typical WTC program, the following distinctions are important:

- ▶ Because the WXTR connection is based on JCA, `ResourceException` exceptions are thrown.
- ▶ Before the program on WXTR is invoked, an `InteractionSpec` object must be created. The program name cannot be passed directly to the `execute()` method.

Although these examples do not cover all application scenarios, the methodologies that are used here can be a good starting point for planning specific migration efforts.

The tasks the WXTR client program performs can be depicted as shown in Figure 4-2.

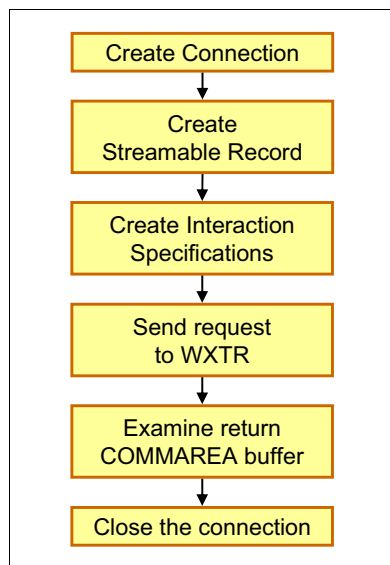



Figure 4-2 Tasks performed by the sample WXTR client program

4.8 Conclusion

In an ideal scenario, Java EE applications that are deployed on one certified application server can be taken exactly as they are and moved to another certified application server. However, this type of migration is almost never the reality. There are myriad reasons for this non-compatibility. WebSphere has acknowledged the problem and is doing something to make it easier to move applications from competitive application servers to WebSphere Application Server.

Although this chapter does not cover all of the permutations and combinations of WTC and Java EE application migrations from Oracle WebLogic to WebSphere Application Server and WXTR, the information presented here should serve as a useful starting point. There are no tools to automatically migrate WTC client applications to WXTR. However, because of the similarity in APIs, migration is not as daunting a task as it might seem.



Modernizing Oracle Tuxedo applications with WXTR

This chapter describes modernization techniques for Oracle Tuxedo applications by using IBM WebSphere eXtended Transaction Runtime (WXTR). It also describes how to use the WXTR Feature Pack for Modernizing Oracle Tuxedo Applications, a tool for migrating Oracle Tuxedo applications to the WXTR environment.

This chapter contains the following sections:

- ▶ WXTR Feature Pack for Modernizing Oracle Tuxedo Applications
- ▶ Advantages of using the WXTR modernization feature pack
- ▶ Using the WXTR modernization feature pack
- ▶ Four-step migration process

5.1 WXTR Feature Pack for Modernizing Oracle Tuxedo Applications

The WXTR Feature Pack for Modernizing Oracle Tuxedo Applications helps you migrate Oracle Tuxedo COBOL and C applications to an IBM WebSphere Application Server environment.

The lightweight feature pack can be installed over an existing WXTR installation. It assists with Oracle Tuxedo application migrations by providing tools for the following phases of the process: analyzing the existing code, building the new applications to work in a WebSphere environment, and deploying the new applications on WebSphere Application Server.

5.1.1 Capabilities

The WXTR Feature Pack for Modernizing Oracle Tuxedo Applications can help migrate client and server COBOL and C applications. The migrated applications then run seamlessly in a WXTR container as part of WebSphere Application Server.

The migrated COBOL and C applications can map Tuxedo record types, error codes, and return values to their WXTR 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 ATMI APIs.

Although the feature pack can port COBOL and C applications with minimal code changes, WXTR provides native interoperability between Java EE and COBOL and C applications. This ability enables seamless interactions between new Java EE programs and the migrated Tuxedo applications.

5.1.2 Components

The WXTR Feature Pack for Modernizing Oracle Tuxedo Applications provides the following tools and libraries to aid in the different phases of application migration:

- ▶ Build tools
- ▶ Migration libraries
- ▶ Java Connector Architecture (JCA) Interface

Figure 5-1 on page 105 shows the components of a typical Oracle Tuxedo-to-WXTR migration scenario by using the WXTR Feature Pack for Modernizing Oracle Tuxedo Applications.

A code analyzer tool also is available in the IBM WebSphere Application Server Migration Toolkit

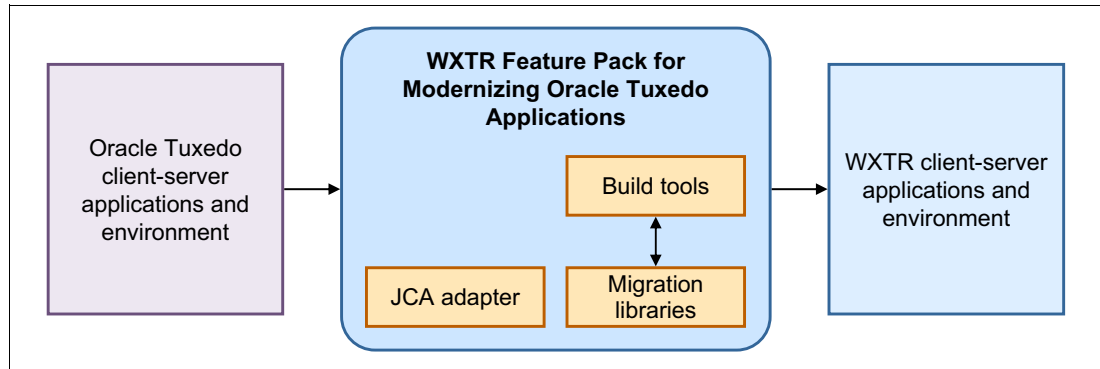


Figure 5-1 Typical components of an Oracle Tuxedo-to-WXTR migration by using the feature pack

Build tools

Tuxedo COBOL and C server applications are compiled by using the **buildserver** tool. Thus, the WXTR Feature Pack for Modernizing Oracle Tuxedo Applications provides a set of interfaces with which users can compile Tuxedo COBOL and C applications without changing existing makefiles. The **buildserver** tool compiles Tuxedo COBOL and C applications by using the feature pack's migration libraries. The tool also generates a loadable module that can run as a WXTR process in a WebSphere Application Server environment.

Migration libraries

The feature pack provides a set of libraries for migrating Oracle Tuxedo functionality to a WebSphere Application Server environment. The **buildserver** tool maps Tuxedo APIs to WXTR APIs in the compile phase. In the link phase, the mapped APIs are linked to corresponding feature pack libraries that act as an interface, which provides data compatibility between the two sets of APIs.

Java Connector Architecture (JCA) Interface

The WXTR Feature Pack for Modernizing Oracle Tuxedo Applications includes a JCA interface for clients to connect to migrated COBOL and C server programs that are deployed on WebSphere Application Server. This JCA interface provides alternative class definitions for the Common Clients Interface (CCI)-supported classes that are defined by Tuxedo. Internally, the WXTR JCA resource adapter is used to make a JCA request from a Java EE environment to COBOL and C environments. By using this connector, Java EE clients that used the Oracle Tuxedo JCA adapter can migrate directly to a WebSphere Application Server environment.

Code analyzer

A code analyzer tool, which is available in the IBM WebSphere Application Server Migration Toolkit for use with IBM Rational Application Developer and IBM Rational Software Architect, is recommended for scanning existing Tuxedo applications to identify the Tuxedo ATMI APIs that were used and whether they are supported in WXTR. This ability makes the analyzer 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.

5.2 Advantages of using the WXTR modernization feature pack

Many companies that invested time and money in developing business logic through COBOL and C-based applications now want to modernize these applications by using Java EE technologies. Not only can they benefit from the wider integration abilities of Java EE, they can also counter skill shortages in niche languages, such as COBOL.

In these situations, the following approaches to application modernization are often the most common:

- ▶ Rewriting existing COBOL or C applications in Java
- ▶ Performing external integrations by using adapters or web services

However, these solutions often provide only basic functionality, and much of the applications' coherence is lost because of their new, more loosely coupled nature.

This circumstance is where the WXTR Feature Pack for Modernizing Oracle Tuxedo Applications can help.

5.2.1 Modernize Tuxedo COBOL and C applications

The WXTR Feature Pack for Modernizing Oracle Tuxedo Applications extends WXTR's core value proposition of hosting COBOL and C applications as a native resource by enabling them to run in a WebSphere Application Server environment. The feature pack benefits companies who have Tuxedo Style Applications and are considering application modernization, such as integration into a service-oriented architecture (SOA).

The feature pack provides the following modernization-related capabilities:

- ▶ Direct functional mapping from Tuxedo ATMI APIs to WXTR APIs, which minimizes costs in terms of effort and time that is required to complete the migration process.
- ▶ A tightly integrated, composite-application serving infrastructure for the migrated COBOL and applications. Also included is the ability of WXTR to host transactional COBOL and C applications, and Java Enterprise applications within WebSphere Application Server.

5.2.2 Reduce costs

Moving to WXTR has inherent benefits that help reduce costs. For companies, the following benefits include:

- ▶ Ability to capitalize on widely available Java EE skill sets to modernize COBOL and C assets
- ▶ Ability to manage multi-language workloads by using the central administration capability of WebSphere Application Server

The feature pack adds value by providing a cost-effective, automated migration solution for Tuxedo-style COBOL and C applications. For more information about the Tuxedo to WXTR migration process, see 5.4, "Four-step migration process" on page 107.

5.2.3 WebSphere Application Server business capability

One of WebSphere Application Server's most important aspects is the ability to integrate seamlessly into an SOA stack. This capability includes the following benefits:

- ▶ Tuxedo COBOL and C applications that are migrated onto WXTR are treated as services that are intrinsically part of an SOA. This configuration simplifies the addition of new SOA services that are written in Java, which can interact directly with deployed Tuxedo COBOL and C services.
- ▶ Applications that include Java and COBOL components can be debugged from the IBM Rational Developer for Power Systems interface. By using this tool, debugging shifts seamlessly between Java and COBOL code.

5.3 Using the WXTR modernization feature pack

WXTR provides access to deployed COBOL and C resources through a JCA interface. This access also is possible by using the WXTR Feature Pack for Modernizing Oracle Tuxedo Applications.

The `WXTRConnector.jar` file that is provided with the feature pack contains classes that resemble Tuxedo CCI classes in form. A servlet program that was written by using the CCI classes of Tuxedo can be compiled to perform the same functions by using the classes that are provided in `WXTRConnector.jar`. The classes also provide ways to define Tuxedo data records, such as string data or byte data, through the `TuxedoStringRecord` and `TuxedoXOctetRecord` classes. Exception handling for a WXTR request is done through the `TPEXception` class.

The advantage of compiling these tuxedo programs with the classes from the feature pack is that it enables unified administration on WebSphere Application Server. The applications can be deployed, monitored, and managed from this single console.

5.4 Four-step migration process

The process to migrate an Oracle Tuxedo application to a WXTR environment can be completed in the following four steps:

1. Use code analysis to select suitable applications for migration.
2. Recompile the client and server applications on WXTR.
3. Deploy the compiled client and server applications on WXTR.
4. Verify the deployed programs.

The migration process is designed to ensure that an application that is running in an Oracle Tuxedo-WebLogic environment can be deployed and then run in a WXTR- WebSphere Application Server environment.

5.4.1 Selecting 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 feature pack. A code analyzer that is included within the IBM WebSphere Application Server Migration Toolkit is an example of tooling that is available to assist with the identification task.

Example 5-1 shows a snippet from a sample Tuxedo ATMI application that is called `SampleOne.cbl`.

Example 5-1 Tuxedo ATMI API usage in `SampleOne.cbl`

```
... .
        CALL "TPSVCSTART"
            USING TPSVCDEF-REC
                TPTYPE-REC
                DATA-REC
                TPSTATUS-REC.
... .
```

The sample program uses the `TPSVCSTART` API. When the code analyzer scans the code, it detects the use of `TPSVCSTART` and marks it as `SUPPORTED` in the consolidated report that is generated, as shown in Example 5-2.

Example 5-2 Code analyzer report for `SampleOne.cbl`

#####				
# Consolidated Report #				
#####				
API NAME	SUPPORTED	UNSUPPORTED	MODIFICATION NEEDED	
TPSVCSTART	1	0	0	

5.4.2 Recompiling client and server applications

The `WXTR` Feature Pack for Modernizing Oracle Tuxedo Applications provides tools to recompile client and server applications in a `WXTR` environment.

Building server applications

Tuxedo ATMI applications that are identified for migration during code analysis are compiled with the `buildserver` tool by using libraries from the feature pack. The compiled modules can then be loaded onto a `WXTR` process to be run. Figure 5-2 shows the Tuxedo server application migration phase.

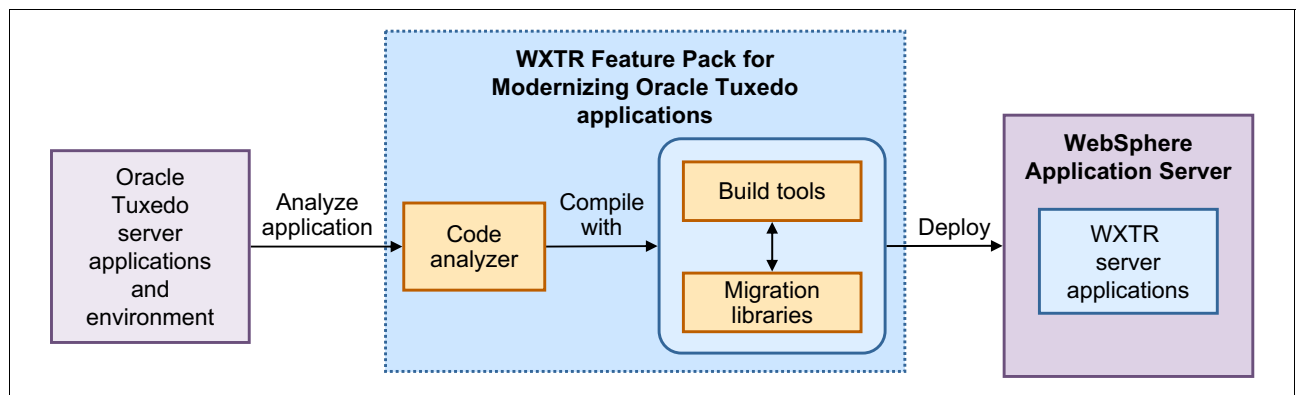


Figure 5-2 Tuxedo server application migration by using code analyzer and build tools

Example 5-3 on page 109 shows the `buildserver` tool’s functionality that uses the sample COBOL application `SampleTwo.cbl`, which contains two Tuxedo APIs: `TPSVCSTART` and `TPCALL`.

Example 5-3 Portions of SampleTwo.cbl program that contains TPSVCSTART and TPCALL APIs

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.    SAMPTWO.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
WORKING-STORAGE SECTION.
```

```
.....
```

```
* Calling TPSVCSTART
```

```
CALL "TPSVCSTART"  
  USING TPSVCDEF-REC  
        TPTYPE-REC  
        DATA-REC  
        TPSTATUS-REC.
```

```
.....
```

```
* Calling TPCALL
```

```
CALL "TPCALL"  
  USING TPSVCDEF-REC  
        ITPTYPE-REC  
        IDATA-REC  
        OTPTYPE-REC  
        ODATA-REC  
        TPSTATUS-REC.
```

```
.....
```

```
* End of SAMPTWO
```

The SampleTwo.cbl program can be compiled by using the **buildserver** tool. By specifying the **-s** option as SAMPTWO, a service that is called SAMPTWO is created. In WXTR terms, this configuration means that a Program Definition was added under the name SAMPTWO in the Program Definition stanza. The **-o** option allows the creation of a COBOL executable file with the wanted name.

Example 5-4 shows how to use the **buildserver** tool from the command prompt, and the associated compilation messages.

Example 5-4 buildserver compilation of SampleTwo.cbl

```
# buildserver -C -s SAMPTWO -o SampleTwo -f SampleTwo.cbl  
PP 5724-V62 IBM COBOL for AIX 4.1.0 in progress ...  
End of compilation 1, program INV-SVC, no statements flagged.  
PP 5724-V62 IBM COBOL for AIX 4.1.0 in progress ...  
End of compilation 1, program SAMPTWO, no statements flagged.  
ERZ094002I/0074: 'buildserver' command completed successfully
```

In addition to the program's .lst, object files, and executable, three files also are created: cics_stub_imat.cbl, cics_stub_imat.lst, and cics_stub_imat.o. The cics_stub_imat.cbl file is an intermediate file that is generated by the **buildserver** tool at every invocation. This file is responsible for transferring control to the requested service routine and is linked with other service files to build an executable file. The .cbl file is deleted upon successful compilation. The .lst file and the object files are retained.

Building client applications

The WXTR Feature Pack for Modernizing Oracle Tuxedo Applications provides the JCA Common Client Interface support to allow Java EE clients to access deployed server COBOL and C programs. WXTR ships with a JCA adapter file, `WXTRConnector.jar`, which contains the WXTR version of Tuxedo CCI classes. WXTR's JCA adapter also provides support for Oracle Tuxedo data records and typed buffers. Figure 5-3 shows the migration process for Tuxedo CCI clients.

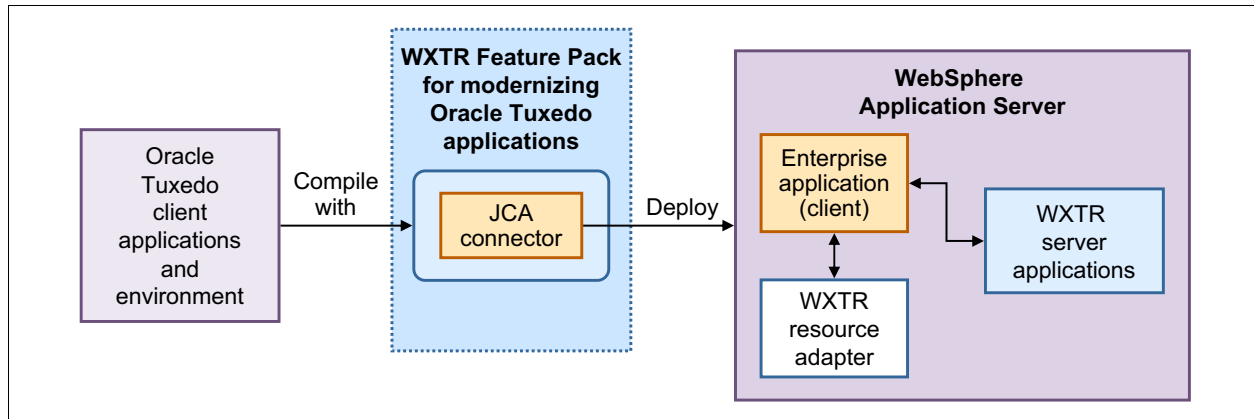


Figure 5-3 Tuxedo CCI Client migration by using WXTR JCA Interface

Java EE applications that are written by using Tuxedo CCI classes can also be directly migrated to a WebSphere Application Server environment by using the JCA adapter. Example 5-5 shows a portion of a sample Java EE client program, `TuxedoStringClient.java`, that was written by using Tuxedo CCI classes.

Example 5-5 A Sample Tuxedo CCI program, `TuxedoStringClient.java`

```
.....
.....

TuxedoStringRecord input = new TuxedoStringRecord();
TuxedoInteractionSpec ccispec = new TuxedoInteractionSpec();
TuxedoJCAConnection tuxc=(TuxedoJCAConnection)tuxcf.getConnection; //Let's
assume tuxcf is a TuxedoConnectionFactory object.
TuxedoInteractionSpec tuxispec = new TuxedoInteractionSpec();
try {
    tuxispec.setFunctionName(programName); // programName is a WXTR resource that
    will be called
    tuxispec.setInteractionVerb(InteractionSpec.SYNC_SEND_RECEIVE);
    catch (TPEException e1) {
        System.out.println("Something went wrong"+e1.getTperrno());
    }
    .....
    try {
        TuxedoInteraction tuxi = (TuxedoInteraction) tuxc.createInteraction();
        tuxi.execute(ccispec, input, output);
    } catch (ResourceException e) {
        e.printStackTrace();
    }
    .....

```

To compile `TuxedoStringClient.java` with the feature pack's classes, we must ensure that the `WXTRConnector.jar` file that is supplied with the WXTR Feature Pack for Modernizing Oracle Tuxedo Applications is in the CLASSPATH. A WXTR client application can be a Servlet or an Enterprise Java Bean (EJB), which is built with the `WXTRConnector.jar` file.

Complete the following steps to compile a servlet application with `CCWConnector.jar` on Rational Application Developer, as described in 2.4, "Modernizing applications" on page 36:

1. Ensure that the `WXTRConnector.java` file is in the `WEB-INF/lib` folder. In Rational Application Developer, add the library by clicking the project that contains the source and selecting **File** → **Properties** → **Add External JARs**.
2. Import the Tuxedo classes, such as `TuxedoInteractionSpec` and `TuxedoInteraction`, from the feature pack's library to the Tuxedo CCI client program, as shown in Example 5-6.

Example 5-6 Importing classes from the feature pack

```
import com.ibm.wxtr.jca.TuxedoJCAConnection;  
import com.ibm.wxtr.jca.TuxedoConnectionFactory;  
import com.ibm.wxtr.jca.TuxedoManagedConnectionFactory;  
import com.ibm.wxtr.jca.TuxedoInteractionSpec;  
import com.ibm.wxtr.jca.TuxedoInteraction;  
import com.ibm.wxtr.jca.TuxedoStringRecord;
```

3. Rational Application Developer automatically compiles the source files, depending on the preference settings.
4. Export the project as a Web Application Resource (WAR) file by clicking the project and selecting **File** → **Export** → **WAR File**. The new WAR file can now be deployed on WebSphere Application Server from the administrative console or directly from Rational Application Developer.

5.4.3 Deploying compiled client and server applications

After the Oracle Tuxedo programs are compiled, the next step in the migration process is to deploy the compiled applications in a WXTR environment.

Server applications

Compiling the programs with the feature pack's **buildserver** tool not only creates loadable COBOL or C modules, it adds the Program Definition and Transaction Definition to the WXTR stanza files. In addition, after creating the loadable modules, **buildserver** tool copies the `ibmcob` file to the WXTR `/bin` directory, where it is loaded onto a WXTR process when it is run.

The WXTR Feature Pack for Modernizing Oracle Tuxedo Applications requires WXTR to be restarted each time a new service is added. Use the following commands to complete this task:

- To stop the server:
`WAS PROFILE PATH/bin/stopServer server`
- To start the server:
`WAS PROFILE PATH/bin/startServer server`

Client applications

The client application can be deployed on WebSphere Application Server as an enterprise application in the form of an Enterprise Archive (EAR), EJB, or WAR file. Likewise, the application can be installed on WebSphere Application Server by using the administrative console or wsadmin scripts.

For more information about deploying enterprise applications, see the WebSphere Application Server Information Center at this website:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=%2Fcom.ibm.was.sphere.express.doc%2Finfo%2Fexp%2Fae%2Ftrun_appl.html

5.4.4 Verifying deployed programs

After the server programs are deployed in the WXTR container and the client programs are deployed as enterprise resources on WebSphere Application Server, the migration process is almost complete. The last step is to verify that the migration succeeded through the use of various log and trace files that are supplied with WXTR and the WXTR Feature Pack for Modernizing Oracle Tuxedo Applications.

The following options are available to monitor the application and verify the successful migration:

► Feature pack server log

You can monitor the execution of a migrated server program that is deployed on WXTR by using the feature pack's server log.

To enable the server log, the WXTR container's Resource Definition (RD) stanza must be configured. This configuration can be done by setting the variable `ImatSrvLog` to "1" by using the following wsadmin commands:

```
AdminTask.cics('[-command update -nodeName myhostNode01 -serverName server1  
-c rd -properties [[ImatSrvLog 1]]]')
```

```
AdminConfig.save()
```

The server log can be disabled by using the following commands:

```
AdminTask.cics('[-command update -nodeName myhostNode01 -serverName server1  
-c rd -properties [[ImatSrvLog 0]]]')
```

```
AdminConfig.save()
```

After the log is enabled, a file called `CSRVL.out` is created in the data directory of the WXTR container. This log contains information about the entry and exit of each Tuxedo API in the migrated program, which is often the memory address of the data passed to the API, any errors that were encountered, and so on.

► Feature pack event log

The WXTR Feature Pack for Modernizing Oracle Tuxedo Applications 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 using these wsadmin commands:

```
AdminTask.cics('[-command update -nodeName myhostNode01 -serverName server1  
-c rd -properties [[ImatUsrLog 1]]]')
```

```
AdminConfig.save()
```

To disable the event log, use these commands:

```
AdminTask.cics('[-command update -nodeName myhostNode01 -serverName server1  
-c rd -properties [[ImatUsrLog 0]]]')
```

```
AdminConfig.save()
```

Upon enabling the feature pack's event log, a file called CULOG.out is created in the data directory of the WXTR container. With this event log, users can include diagnostic messages to verify the successful execution of an application.

► **WebSphere Application Server logs**

In addition to verifying the execution of migrated server applications, it is possible to verify the behavior of migrated client applications by monitoring the following WebSphere Application Server logs:

- `SystemErr.log`: This log contains information about Java exceptions and stack traces. These exceptions often are caught in enterprise applications.
- `SystemOut.log`: This log contains messages generated by applications in WebSphere Application Server. the log also contains error messages that are not classified as errors by the operating system. The log also records if a migrated application fails to start or stop.

5.4.5 Four-step checkpoint for migration

The four-step process for migrating Oracle Tuxedo applications is shown in Figure 5-4 on page 114.

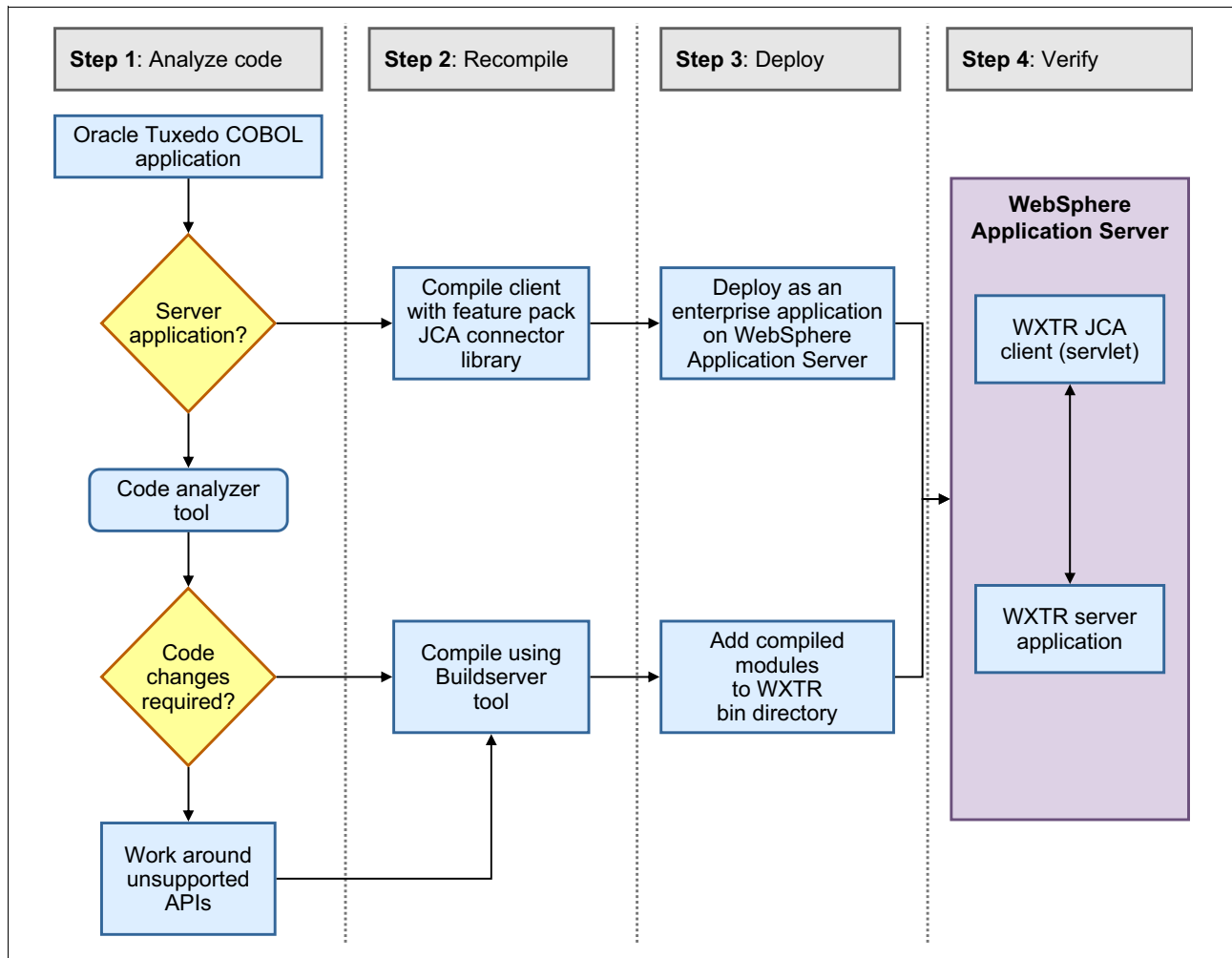


Figure 5-4 The four-step migration process

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

IBM Redbooks

The following IBM Redbooks publications provide more information about the topic in this document. Some publications referenced in this list might be available in softcopy only.

- ▶ *WebSphere Application Server V8.5 Migration Guide*, SG24-8048
- ▶ *Rational Application Developer for WebSphere Software V8 Programming Guide*, SG24-7835
- ▶ *WebSphere Application Server V7.0 Security Guide*, SG24-7660

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, draft, and additional materials, at this website:

<http://www.ibm.com/redbooks>

Online resources

The following websites also are relevant as other information sources:

- ▶ WXTR Information Center
<http://pic.dhe.ibm.com/infocenter/wxtrform/v2r1/index.jsp>
- ▶ WebSphere Application Server Information Center
<http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp>
- ▶ Rational Developer for Power Systems Software
<http://publib.boulder.ibm.com/infocenter/iadthelp/v8r0/index.jsp>
- ▶ Using WebSphere Developer for zSeries V6 to Connect to COBOL/CICS
http://www.ibm.com/developerworks/websphere/library/techarticles/0509_barosa/0509_barosa.html
- ▶ TXSeries for Multi-Platform
<http://www-01.ibm.com/software/http/cics/txseries/>
- ▶ Rational Application Developer
http://publib.boulder.ibm.com/infocenter/rtnlhelp/v6r0m0/index.jsp?topic=%2Fcom.ibm.rational.rad.books%2Ficwelcome_product_rad.htm
- ▶ CICS Transaction Gateway
<http://www-01.ibm.com/software/http/cics/ctg/>
- ▶ IBM WebSphere Migration Toolkit Migration Forum
<http://www.ibm.com/developerworks/forums/forum.jspa?forumID=2106>

- Application Migration Tools for Competitive Migration

http://public.dhe.ibm.com/software/dw/wes/migrationtoolkit/ApplicationMigrationTool_en_US.3.0.0.pdf

Help from IBM

IBM Support and downloads

<http://ibm.com/support>

IBM Global Services

<http://ibm.com/services>



Modernizing Applications with WebSphere eXtended Transaction Runtime



Improve productivity across mixed-language applications

Optimize existing C and COBOL assets

Manage CICS-style data with DB2

IBM WebSphere eXtended Transaction Runtime V2.1 is an addition to the IBM Transaction Processing capabilities. This product provides a fast, scalable, and reliable transaction processing experience. Many customers have invested much time and effort in the development of business logic in CICS style COBOL and C applications and are looking to unlock the value of those applications and extend them by using Java EE.

This paper helps you explore this product and provides information that helps you host your CICS style COBOL and C applications on a WebSphere platform. This paper also provides you with a detailed step-by-step approach for modernizing your existing Tuxedo-based applications through a migration to WXTR.

This paper is intended for developers and architects who want to extend and reuse their CICS style COBOL and C applications.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks