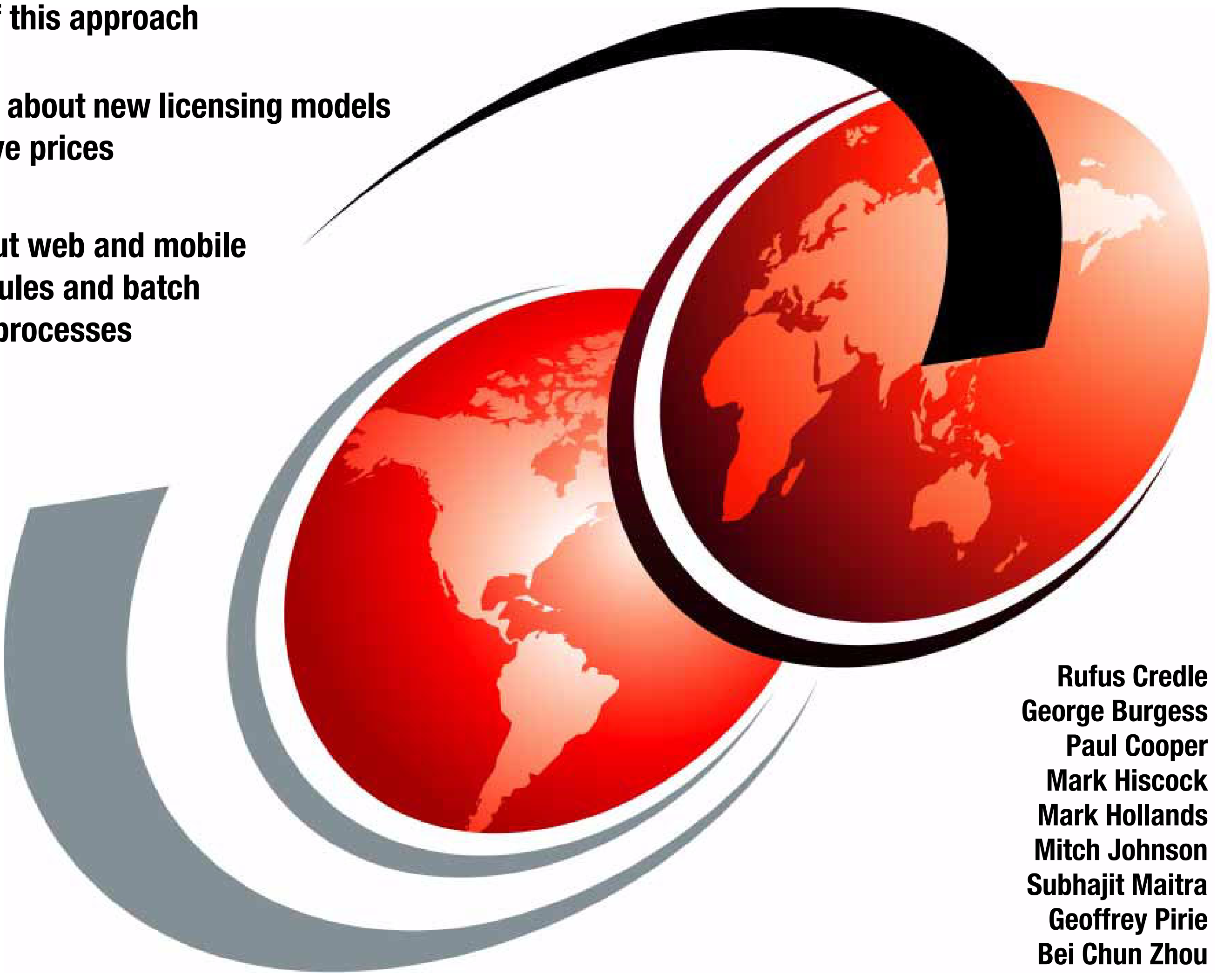
A Software Architect's Guide to New Java Workloads in IBM CICS Transaction Server

Review the architectural and technical benefits of this approach

Get details about new licensing models at attractive prices

Learn about web and mobile business rules and batch workload processes



Rufus Credle
George Burgess
Paul Cooper
Mark Hiscock
Mark Hollands
Mitch Johnson
Subhajit Maitra
Geoffrey Pirie
Bei Chun Zhou

Redbooks

ibm.com/redbooks

International Technical Support Organization

A Software Architect's Guide to Java Workloads in IBM CICS Transaction Server

December 2014

SG24-8225-00

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

First Edition (December 2014)

This edition applies to IBM CICS Transaction Server for z/OS, IBM WebSphere Application Server Liberty Profile, WebSphere Application Server for z/OS, and IBM Operational Decision Manager for z/OS.

Contents

Notices

Trademarks

Preface

This IBM® Redbooks® publication introduces the IBM System z® New Application License Charges (zNALC) pricing structure and provides examples of zNALC workload scenarios. It describes the products that can be run on a zNALC logical partition (LPAR), reasons to consider such an implementation, and covers the following topics:

- Using the IBM WebSphere® Application Server Liberty profile to host applications within an IBM CICS® environment and how it interacts with CICS applications and resources
- Security technologies available to applications that are hosted within a WebSphere Application Server Liberty profile in CICS
- How to implement modern presentation in CICS with a CICS Liberty Java virtual machine (JVM) server
- How to share scenarios to develop Liberty JVM applications to gain benefits from IBM CICS Transaction Server for z/OS® Value Unit Edition
- Considerations when using mobile devices to interact with CICS applications and explains specific CICS technologies for connecting mobile devices by using the z/OS Value Unit Edition
- How IBM Operational Decision Manager for z/OS runs in the transaction server to provide decision management services for CICS COBOL and PL/I applications
- Installing the CICS Transaction Server for z/OS (CICS TS) Feature Pack for Modern Batch to enable the IBM WebSphere batch environment to schedule and manage batch applications in CICS

This book also covers what is commonly referred to as plain old Java objects (POJOs). The Java virtual machine (JVM) server is a full-fledged JVM that includes support for Open Service Gateway initiative (OSGi) bundles. It can be used to host open source Java frameworks and does just about anything you want to do with Java on the mainframe. POJO applications can also qualify for deployment using the Value Unit Edition. Read about how to configure and deploy them in this companion Redbooks publication:

IBM CICS and the JVM server: Developing and Deploying Java Applications, SG24-8038

https://www.redbooks.ibm.com/redbooks.nsf/RedpieceAbstracts/sg248038.html?Open

Examples of POJOs are terminal-initiated transactions, CICS web support, web services, requests received via IP CICS sockets, and messages coming in via IBM WebSphere MQ messaging software.

Authors

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

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

George Burgess is a Software Engineer in CICS Development at IBM Hursley Park in the UK. Previously, he spent three years as a subject matter expert on CICS Transaction Server on z/OS for the Peoples Republic of China. He has 29 years of experience as an application programmer, systems programmer, CICS developer, and IBM Tivoli® OMEGAMON® XE for CICS developer. His other areas of expertise include Common Business Oriented Language (COBOL), IBM DB2® databaes, Java, IBM MQ, IBM IMS™ database management system, Data Language Interface (DL/1), Virtual Storage Access Method (VSAM), JCL,
IBM z/OS, and OMEGAMON.

Paul Cooper has worked in the CICS Development organization at IBM Hursley Park for more than15 years. In that time, he has helped develop the Java technology in CICS, web services in CICS, mobile support for CICS, and the CICS cloud infrastructure.

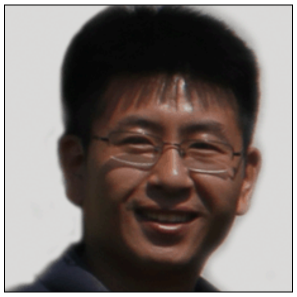
Mark Hiscock is the Operational Decision Management development team lead for IBM z/OS. He is based in Hursley, in the UK. He joined IBM in 1999 and holds a first-class degree in Computer Science from the University of Portsmouth. He has more than 10 years experience working in mainframe development on products such as IBM Operational Decision Manager, MQ, CICS, DB2, WebSphere Message Broker, and WebSphere Application Server.

Mark Hollands is a Software Engineer in the CICS Development organization. He has two years of experience in developing IBM CICS Explorer® plug-ins for several of the CICS tools, including CICS Configuration Manager and CICS Deployment Assistant. After graduating with a degree in Computer Science, Mark joined IBM as a Software Developer. He worked on WebSphere Voice Response before transferring to CICS Tools Development.

Mitch Johnson is a Client Technical Specialist for IBM MQ, IBM Operational Decision Manager, and CICS in Advanced Technical Skills in the United States. He was previously a consultant for WebSphere in IBM Software Services, where his expertise was WebSphere Application Server on z/OS. He has worked on five previous IBM Redbooks publications and Redpapers that dealt primarily with connectivity to z/OS resources from WebSphere Application Server and other resources. His areas of expertise include CICS, DB2, IMS, MQ, z/OS, Java, and Java Platform, Enterprise Edition connectors and security.

Subhajit Maitra is a Senior IT Specialist and member of the IBM North America System z WebSphere Technical Sales team. His expertise includes IBM Operational Decision Manager, IBM Integration Bus, and IBM MQ on System z. Subhajit is also an IBM Global Technical Ambassador for Central and Eastern Europe, where he helps IBM clients implement business-critical solutions on IBM System z. His 21-year career in information technology includes roles as a developer, designer, and architect in the healthcare, financial services, and government industries. Subhajit previously worked with the ITSO in creating and delivering workshops worldwide and as a co-author of previously published Redbooks. He holds a master's degree in computer science from Jadavpur University, in Kolkata, India, and is an IBM zChampion.

Geoffrey Pirie is a CICS product marketing manager with the IBM Software Group for application and integration middleware, at IBM Hursley Park in the United Kingdom. He joined IBM as a developer and later moved into management of the CICS System Test and Performance team. His experience and growth led him to join the marketing team for CICS, where he helps manage the CICS portfolio.



Bei Chun Zhou is Software Engineer on the IBM CICS Transaction Server for z/OS service team. He graduated from Tsinghua University and then earned a master's degree from the Chinese Academy of Sciences. He has worked at IBM for four years as a CICS L3 support representative and is now the lead of China CICS L3 team. His expertise is mainly in the CICS Transaction Server. He supports IBM CICS clients in problem solving, system upgrades, health checks, performance tuning, and new feature enablement.

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

Tamikia Barrow, Debbie Willmschen
International Technical Support Organization, Raleigh Center

Amanda Ballard-Stuart, Staff Administration Assistant
IBM Hursley

Matthew Wilson, Ivan Hargreaves, Daniel Millwood, Michael Wang, Ian Mitchell
IBM Development
IBM Hursley

Andy Bates, Adrian Kyte
IBM Marketing and Project Management
IBM Hursley

Now you can become a published author, too

Here is 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:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us.

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

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

  ibm.com/redbooks

- Send your comments by email:

  redbooks@us.ibm.com

- Mail your comments:

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

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks
- Follow us on Twitter:

  http://twitter.com/ibmredbooks
- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm
- Stay current on recent Redbooks publications with RSS Feeds:

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

Part 1

New workloads on the mainframe

In this part, we describe mainframe workload pricing and help you understand what will qualify for the IBM System z New Application License Charge and Value Unit Edition.

1

Mainframe workload pricing

This chapter introduces the IBM System z New Application License Charge (zNALC) pricing structure. It also gives examples of zNALC workload scenarios, which are the focus of this book. For more information, see the IBM System z Software Pricing web page:

http://www.ibm.com/systems/z/resources/swprice

This chapter covers the following topics:

1.1 Advantages of the Value Unit Edition pricing model

For many organizations, the ongoing pressure to manage long-term operational costs can be a psychological barrier when considering deployment of new applications to IBM System z. Yet System z is still one of the most capable and cost-effective platforms for running mixed workloads, with the qualities of service on which these organizations depend. In recognition of these ongoing pressures, IBM has introduced the Value Unit Edition (VUE) pricing model to provide clients with greater flexibility to balance the capital and operating expenses of new projects.

System z New Application Licence Charges (zNALC) gives organizations the opportunity to run a new workload on System z at a reduced cost if that workload is a qualified “new workload” application. An example of a qualified application is a Java language business application running in IBM CICS Transaction Server for z/OS Value Unit Edition (CICS TS VUE) or IBM WebSphere Application Server.

Running CICS Transaction Server for z/OS using the Monthly License Charge (MLC) pricing option is normally considered an operating expense (OpEx). This is because the expenses are charged monthly and are directly related to the CPU use for the month. However, IBM CICS Transaction Server for z/OS Value Unit Edition is a one-time-charge license, so it is categorized as capital expenditure (CapEx). This one-time-charge license can run in a reduced price zNALC logical partition (LPAR). The same principle applies to other qualifying software that is required to support the application.

CICS Transaction Server is part of a family of products that are available as a Value Unit Editions. Other IBM software that is available as Value Unit Editions include IBM DB2 for z/OS, IBM IMS database, IMS Transaction Manager, and IBM WebSphere MQ for z/OS.

1.2 CICS Transaction Server Value Unit Edition benefits

CICS Transaction Server Value Unit Edition (VUE) allows you to deploy new Java workloads at a fixed cost. zNALC provides a reduced price for z/OS. For clients who have an IBM System z Application Assist Processor (zAAP), Java workload can be transferred to the specialty processor, further reducing the cost of running new Java workloads under CICS Transaction Server.

CICS Transaction Server Value Unit Edition contains a version of the WebSphere Liberty Profile at no extra charge, running in the highly scalable JVM server architecture within CICS. A separate WebSphere Application Server license is not required to run WebSphere Liberty Profile applications within CICS TS. Deploying new Java workload to Liberty on CICS allows you to use many of the Liberty capabilities as well as integrating Liberty applications with the JCICS API, providing the best of both worlds in a single managed runtime.

The IBM JVM uses the latest hardware advances in System z including features such as Transactional Memory (for improved concurrency in multithreaded applications) and large 1MB pageable memory using flash. An IBM benchmark shows 40% improvement in Java workloads running on zEC12 using Java7SR3 using hardware improvements compared to z196. CICS TS V5 supports Java7SR1 or later, which enables you to use the most efficient JVM runtime available on System z.

1.3 Business value

The Value Unit Edition (VUE) family of products provides a choice for how to budget for new projects. Previously, all new projects would primarily result in an increase in operational expenditure. Deploying to a VUE environment on a zNALC LPAR enables a proportion of the project cost to be a capital expense. In organizations where strict budgetary controls are in place, having the flexibility to choose between these two budgets can help to get new projects financed and delivered.

You can deploy VUE versions of products to use new capabilities without needing to migrate your existing installations to the latest software levels. For example, CICS TS VUE can interoperate with an existing version of CICS TS (MLC) at a lower level, without triggering the single-version-charging (SVC) window. Furthermore, you can start by deploying the CICS TS Developer Trial, a free version of CICS that allows you to evaluate product capabilities, and later upgrade it, in place, to either CICS TS VUE or CICS TS (MLC). There is no need to uninstall and reinstall software.

1.4 Why Java works on the mainframe

The idea that Java and the mainframe are a bad pairing is outdated and inaccurate. If you still hold the opinion that the two do not fit well together, it is time to reconsider.

Java is a language, and there are many to choose from, but not all languages get the same investment that Java has had, particularly on the mainframe. Perhaps more importantly, Java is also a runtime platform. Java on System z has received significant and ongoing investment from IBM, which has built the Java virtual machine for System z from the ground up, developed hardware that can be used by the JVM, and built core middleware technology using Java. This makes Java on the mainframe one the world’s most optimized Java environments.

System z is something special in itself. There is no other machine that has such a vast heritage with such a profound effect on business, science, and people, from helping to put man on the moon, to becoming the heart of the world’s financial system, or to buying cosmetics with a mobile phone from the comfort of a chair. The mainframe has continued to evolve to incorporate the latest advances in technology to deliver the most capable platform the IT industry has.

It's not surprising to find System z at the heart of the 24x7 always-on world that we have come to expect, processing millions of business transactions every day. IBM estimates that 80% of the world’s corporate data resides or originates on the mainframe. So you might say that it represents the “center of gravity” for business applications and data.

Java’s popularity as a language for enterprise systems has grown steadily. Despite now being around 20 years old, *eWeek* ranked Java number 1 in its Top 10 Programming Languages for Job Seekers in 2014.[1] It is a well-structured language, suitable for writing clean and robust code, where the developer can focus on the problem, take advantage of the language features, and write something that a colleague will understand decades later, using the same tools that are used for writing code on distributed platforms.

Java also benefits from being the language of choice for many universities around the world, where it forms the basis of computer science and software engineering courses. Graduates

[1] Darryl K. Taft, “Top 10 Programming Languages for Job Seekers in 2014,” eWeek, February 2014 http://www.eweek.com/developer/slideshows/top-10-programming-languages-for-job-seekers-in-2014.html

understand it, have been trained to use it, and know its strengths. Java has an established ecosystem, which means it won't be going out of fashion any time soon. Consequently, there are a huge number of open source projects that provide frameworks and services for all kinds of tasks. The number of Java programmers that exist exceeds nine million.

Plus, it performs. There might be people who can write tight assembler code to perform a very specific function faster than the equivalent COBOL or Java, but who can maintain it and for how long? The IBM Just in Time (JIT) compiler progressively optimizes Java code at runtime. Although our hypothetical assembler guru might be able to do a good job in a specific case, the JIT compiler considers all of it and optimizes code for the hardware on which it's running. As IBM invests in hardware improvements on System z, it enhances the JIT compiler to take advantage of them. The result is continued performance improvements for Java applications as IBM updates the JIT compiler to produce more efficient code at runtime.

Some might say that Java and the mainframe didn't get off to the best start, and it is certainly true that some people formed an opinion in the early years that the two were not suited to each other. You can still find these opinions in old blog posts and forums on the Internet, but they do not reflect the current state of Java on the mainframe.

It is not entirely surprising that a fledgling technology, as Java obviously was when it first came to the mainframe, did not perform as well as the longstanding applications that it was compared to, which were written in COBOL. Those existing applications had received years of progressive refinement and refactoring, perhaps even decades of it, so the comparison was unfavorable and some people formed a negative opinion by focusing on short-term comparisons.

Today's Java on System z runs in the IBM 64-bit JVM, uses features such as hardware transactional memory, large memory pages, and hardware instructions that go back to the IBM z990 that was introduced in 2003. It has more than a decade of investment and refinement behind it. The JVM and JIT have both been enhanced in step with Java specification developments, and the hardware has had capabilities added specifically to enhance Java workloads. IBM also introduced the z Application Assist Processor (zAAP) that runs Java workloads on dedicated processors, significantly reducing the long-term operating cost of Java workload on the mainframe.

Comparing Java today to the Java of 10 years ago is like comparing Formula 1 racing cars that are a decade apart and asserting that the technology has not changed much in the passing years. It's not really a comparison that is worth making because they are simply not the same thing.

1.5 When and where to put Java on System z

Deciding whether System z is the right place to put new Java applications has not always been the simple decision that people would like it to be. Depending upon whom you ask, the answer can be anything from "Put everything on z" to "never" to "it depends," with the last answer typically accompanied by a long technical discussion of the pros and cons. None of these answers are particularly helpful. The first leaves you wondering whether the person really understands why you have asked the question, the next leaves you wondering whether technology alone can provide the answer, and the last often feels like a passionate and unsubstantiated opinion.

Today, if you are asking yourself where the best place to deploy a new Java application is, you should probably first ask yourself, "Where is the center of gravity of the existing applications and data that this new Java application will interact with?"

You might wonder what we mean when we use “center of gravity” in the context of technology. Imagine a typical IT environment where there are lots of disparate systems that need to interact with each other. Now consider what the typical pattern of interactions will be between the proposed new Java application and those existing systems, applications, and data. The center of gravity is wherever the greatest concentration of interactions is. Positioning the new Java application on or near the center of gravity minimizes the architectural complexity of the solution, which in turn has a positive impact on things such as response time, maintenance costs, and resiliency.

Consider a simple example. Imagine that you are planning to build and deploy a new Java application. The new application will provide some entirely new capabilities but will need to interact with the existing business services hosted in CICS that modify data hosted on the mainframe. These services provide the business function that you need and they also embody preferred practices and comply with your audit and governance policies. So being able to reuse them is essential.

By applying the center of gravity principle, you can quickly assess the level of interaction that your new application will have with the existing mainframe solution. In this example, the center of gravity is clearly in CICS on the mainframe. So what Java environment can you expect to find there? CICS Transaction Server (TS) version 5 supports the IBM 64-bit JVM, so it provides a multithreaded JVM server environment directly in CICS TS. If your application could benefit from some of the capabilities of the IBM WebSphere Application Server Liberty Profile, for example a presentation layer built using JavaServer Pages (JSPs) and servlets, you will find that the Liberty Profile is embedded in the CICS JVM server and packaged without charge as part of CICS TS V5.

Another example is a new Java application that interacts with one or more non-mainframe systems, such as a distributed IBM DB2 database, perhaps in combination with a third-party packaged application that us running in an IBM SoftLayer® public cloud. If there is little or no requirement for your new Java application to have access to applications or data on the mainframe, the center of gravity for that application is clearly not on the mainframe. In this example, the question to ask might be whether the center of gravity is nearer the distributed DB2 database or the cloud-based third-party application. Either way, WebSphere Application Server has deployment options in both of these environments.

The center of gravity principle works because no matter where the center of gravity is, there is an enterprise-grade Java environment there to accompany it. The goal of the center of gravity principle is not to define an absolute answer, because there might be more considerations that steer you in another direction. The goal is to consider options that are congruent with past business decisions and that recognize the broad range of Java capabilities available to you, regardless of their locations.

1.6 Defying gravity

Although the center of gravity principle helps to keep your decision-making process grounded, there are some specific cases where following the principle without question might not be optimal. The most likely case is in a mixed platform environment, where both System z and distributed servers co-exist. In mixed platform environments, the center of gravity might appear to be in the distributed server environment, even if it has interactions with System z based services. So the logical conclusion would be to locate a new Java application somewhere in the distributed environment. Is this the right answer?

To be clear, we are not suggesting that this is a wrong answer, but it is worth stopping to consider the economics of growing a distributed server environment, particularly when

System z is also present. The case for consolidation onto System z, in particular Linux for System z, is strong, because it breaks the economic pattern of incremental costs associated with each new distributed server.

Linux on System z virtualization enables new instances to be added to the environment quickly and easily, without needing to worry about floor space, power, cooling, cabling, physical maintenance, and much of the cost associated with the introduction of a physical device. Moreover, if your organization is motivated by the economics of a cloud environment but concerned about migrating sensitive data to the cloud, the combination of Linux on System z and z/OS can form the basis of an efficient private, hybrid cloud environment.

The economic argument extends further. In the same way that System z provides zAAP processors dedicated to running Java workloads, it also provides the Integrated Facility for Linux (IFL) processors. These are dedicated to running Linux on System z, which provides cost and scalability benefits. Consolidation onto System z also offers advantages from the collocation of applications and data, reducing network latency by taking advantage of System z IBM HiperSockets™, and simplifying the application architecture. The IBM z/VM® operating system provides an efficient virtualization environment that enables you to reach nearly 100% processor use, even in mixed workload environments. This reduces the cost per transaction.

Managing costs, particularly operational expenses, is a top priority for all organizations. Consolidating distributed server environments can directly reduce operational expenses, but if the economics for consolidation don't apply to your environment (perhaps you have already consolidated), there are still options to consider for managing operational expenses on System z when deploying new Java applications to the platform. These options, described throughout the remainder of this book, explain how CICS Transaction Server Value Unit Edition can be used to deploy new Java workloads, using a one-time-charge pricing model.

1.7 Solution overview

This IBM Redbooks publication provides a set of technology-lead scenarios that demonstrate how CICS Transaction Server Value Unit Edition can be deployed to address some typical business challenges, ranging from mobile enablement of existing applications to the development of a modern, hybrid batch environment. All of these examples share two common traits: They are new workload scenarios and they can be deployed to CICS TS VUE.

1.7.1 Using the Liberty profile to modernize interfaces

We begin with a review of the WebSphere Application Server Liberty Profile (Liberty), which is packaged with CICS Transaction Server, and consider how Liberty can be used to modernize existing interface technologies. Liberty provides capabilities such as JSPs and servlets, which offer developers familiar who are with web technologies the capability to build rich web-based interfaces that interact with existing CICS applications and services. For many clients, these backend applications represent the cumulative investment in business process, governance, and audit requirements, and they represent core business value in the form of intellectual property and competitive advantage. For many, these are irreplaceable, so providing new methods to interact with these CICS TS services enables businesses to continue to benefit from their investments yet do so using modern interfaces, languages, and capabilities that are available in the Liberty profile.

The following chapters explain more about the Liberty profile:

- Chapter 2, “Introduction to the Liberty JVM server” on page 15, describes configuring Liberty in CICS.
- Chapter 3, “Using CICS Liberty JVM servers to develop application interfaces” on page 25, covers approaches to modernizing interfaces.
- Chapter 4, “Porting JEE applications to a CICS Liberty JVM server” on page 33, describes consolidating Liberty applications in CICS TS.

1.7.2 Optimizing mobile workloads to connect with customers and employees

Part 3, “Mobile devices” on page 45, addresses the growth of mobile technologies that require new mobile workloads. This has been well described in the press. For many, it is the primary driver of growth and new development. Building mobile solutions requires so much more than simply building the mobile application, because a mobile application must meet the expectations of users who live in the always-on, always current, world of connected mobile devices.

Business process optimization for both business-to-consumer (B2C) and business-to-employee (B2E) are dominant factors in the next generation of mobile application development as business moves beyond simple interaction using a mobile device into the realm of the integrated mobile channel. In both of these cases, the value of the mobile application is not simply that it's on a mobile device but that it is integrated with the corporate systems, data, and processes that the business has built to offer competitive services to its customers. When these systems reside on the mainframe, it is natural to want to use them.

The Mobile section explains how CICS TS can provide mobile enablement of existing backend services. It begins by considering the capabilities offered by Liberty, such as JAX-WS and JAX-RS for XML web services and RESTful web services, respectively, before considering z/OS Connect and later options for connecting to Java applications by using the data transformation services of CICS.

1.7.3 Building timely, scalable decisions into software

Part 4, “IBM Operational Decision Manager” on page 71, is also relevant to our always-on, always-connected world, where it becomes increasingly important to make the right decision at the right time. Essential to successful decision management is being able to make decisions in a timely fashion and to locate the decision-making process close to the systems that are dependent upon them. In times gone by, the decision process could be a human task, but these days, efficient decision-making process are codified and embedded in software components that are accessible from across the enterprise. IBM Operational Decision Manager provides the capabilities necessary to codify, manage, update, and deploy decisions, and it can be hosted in CICS TS. This enables you to build a highly scalable and robust decision management solution that is located with critical enterprise workloads.

For more information about decision management and the architectural options for deploying Operational Decision Manager, read Chapter 8, “Decision management integrated in IBM CICS Transaction Server” on page 73. For details about deploying it to CICS TS, see Chapter 9, “Implementing decision management in CICS TS” on page 85.

1.7.4 Updating batch processing

The last section, Part 5, "Modern Batch feature" on page 95, covers modern batch processing. It explains how you can use new technology to reduce the size of your batch window and transform your batch processing and OLTP workloads to make them more cooperative. The traditional batch window, where online systems are shut down, is shrinking as the need for always-on processing increases. Yet the two approaches to data processing are, in their traditional forms, mutually exclusive. By using WebSphere Java Batch and the CICS Feature Pack for Modern Batch, you can build a hybrid batch environment that allows for both methods to operate together, where online applications remain online longer, and new or existing batch workloads co-exist and use some of the capabilities that have previously been unavailable to use for batch processing. By using the WebSphere and CICS combination, you have an alternative way of managing and pricing batch workloads.

Chapter 10, "Modern batch workloads" on page 97 reviews batch processing considerations and the inherent benefits that can be gained by building a hybrid batch environment. Chapter 11, "Modern batch use scenario" on page 113, provides an example of how to build and deploy a modern batch application into CICS TS VUE.

1.8 Value Unit Edition incentives and implementation scenarios

Note: IBM CICS Transaction Server for z/OS Value Unit Edition runs on a zNALC enabled LPAR on z/OS. To deploy it to CICS TS VUE, a workload must meet simple criteria to demonstrate that it is a "net new" Java workload, in accordance with zNALC pricing conditions or that it is a qualifying application. Both are described under the zNALC tab of the IBM System z Software Pricing web page:

`http://www.ibm.com/systems/z/resources/swprice/mlc/znalc.html`

Section 1.5, "When and where to put Java on System z" on page 6, described a range of technical capabilities that can be deployed to CICS TS VUE. This list is not exhaustive, and there are many other use scenarios. However, there are a few common reasons that an organization considers deploying new workloads to VUE.

1.8.1 Do more sooner at less cost

For most businesses, a large proportion of what they are doing with their IT budgets can be categorized as solutions that either get things done sooner or cost less. CICS TS VUE provides both, because it reduces the cost of new workloads and provides capabilities that enable you to get things done faster.

For those who are focused on cutting costs, one obvious consideration is consolidation, either of the physical servers or the software architecture components. Both carry a cost. As an example, consider an organization that has progressively developed an application in CICS. Such an organization might have initially developed a terminal-based application, eventually replacing that with a fat-client solution running on distributed servers, perhaps also building some applications that use web technology hosted on web servers and interact with CICS applications.

The concept of consolidation can be applied in many ways to this example, bringing various benefits depending on exactly what it being consolidated. For example:

- Consolidation to reduce architectural complexity and, therefore, maintenance cost
- Consolidation to reduce the operational expense of managing large data centers
- Consolidation to reduce network latency and improve response times
- Consolidation of “legacy” or existing interface layers to increase productivity and introduce new interactions

Before the introduction of CICS TS VUE, these examples, although valid, might have involved challenges when considering the financial factors involved. This no longer has to be the case, because VUE changes the pricing and allows a fixed price for such projects. By using VUE, a client in the circumstances described would be able to benefit from all of these consolidation options. Therefore, they could minimize architectural complexity, remove unnecessary hardware, improve the response time of applications, and use new languages and technology to create rich mobile and web applications.

1.8.2 Do things faster

The second type of IT budget spending is directed toward *increasing service agility* or doing things faster. The latest features of CICS TS, particularly those offered by the inclusion of the Liberty profile and the mobile capabilities of CICS TS, are well suited to organizations that want to build dynamic interactions that use the applications and services in existing CICS applications. VUE enables you to consider building these new workloads by using all of the features of CICS, integrating with existing CICS applications and services that are already in production, yet doing so with a degree of separation and financial security. You can price the project to use capital budgets and rapidly deploy new applications that use the existing skill base of development and operations staff.

Consider another client, again with an investment in CICS applications, who wants to build a “next generation” mobile application to replace their existing mobile solution. In this example, the client has a broad range of services hosted in CICS and wants to provide a simple interface to these services that doesn't require additional servers in the data center, is built by using popular programming models for mobile development, and provides a lightweight interface for operations staff to track and monitor key performance metrics for the application. For this client, improved service agility might include the following elements and benefits:

- RESTful APIs for accessing services from mobile devices, making integration quick and easy for application development
- Java servlets and JSPs to create web-based performance metrics viewable in a web browser, making the deployment of interface updates quicker and centrally managed
- A structured approach to packaging and managing application updates so that operations staff can be confident when deploying new changes, speeding up deployment times

In this example, before CICS TS VUE, a client who saw value in the Liberty, mobile, and applications packaging capabilities of CICS would have needed to migrate their existing CICS installation to the latest level of CICS TS to gain access to these features. But using CICS TS VUE, the client can deploy the latest version of CICS to a separate LPAR, yet connect to all of the applications and services in their existing CICS environment and use the latest CICS capabilities. This offers all of the capabilities of the latest technology and enables the cost of the project to be allocated from the capital budget.

Although these two scenarios are somewhat trivial, they demonstrate that VUE is not simply about changing the price of a project from an operation expense to a capital expense, but that VUE offers a complete CICS solution to many different business problems.

Part 2

Liberty profile and CICS

This part introduces the concepts and features of the IBM WebSphere Application Server Liberty profile within a CICS Transaction Server for an IBM z/OS environment. It describes two key benefits of the CICS Liberty environment, including the modernization of the presentation layer of CICS applications and how consolidation and collocation of presentation and data can be beneficial in a CICS infrastructure.

This section includes the following chapters:

2

Introduction to the Liberty JVM server

This chapter describes the IBM WebSphere Application Server Liberty profile, its ability to host applications within a CICS Transaction Server (TS) environment, and how it interacts with CICS TS applications and resources. It also describes the security technologies that are available to applications hosted within a Liberty profile in CICS TS.

This chapter covers the following topics:

2.1 Evolving application servers

As software development methods develop over time to deliver more functions to customers quickly and reliably, being able to incorporate new technologies into existing environments has never been more important. The ability for an application server like CICS TS to support these continuously changing demands is essential to deliver these new services. The challenges on application servers include:

- The ability to rapidly deploy (and redeploy) application artifacts as part of a DevOps continuous integration process.
- Software should be modular and easily assembled. This allows applications to be rapidly composed from existing modules and easily deployed into the runtime environment.
- Modern application patterns, such as RESTful web services and responsive UI, are rapidly becoming more popular. Application servers must be able to adopt these new programming models.

The WebSphere Application Server Liberty profile (Liberty) is lightweight and easy to install. It is used to develop and deploy applications. Therefore, it provides a convenient and capable platform for developing and testing your web and OSGi applications. Liberty is built by using OSGi technology and concepts. The fit-for-purpose nature of the runtime relies on the dynamic behavior inherent in the OSGi framework and service registry. As applications (bundles) are installed or uninstalled from the framework, their services are automatically added or removed from the service registry. The result is a dynamic, composable run time that can be provisioned with only what your application requires and responds dynamically to configuration changes as your application evolves.

CICS Transaction Server for z/OS (CICS TS) V5.1 added support for Liberty to run within a Java virtual machine (JVM) server in CICS TS. This has been extended in CICS Transaction Server for z/OS V5.2, which added more supported features for web applications running in a Liberty JVM server within CICS TS.

2.2 Advantages

Liberty is a simple, lightweight development and application runtime environment that offers these benefits:

- *Simple to configure:* Configuration is read from a single XML file (`server.xml`). CICS TS extends this simplicity further by adding the ability to automatically generate this XML file.
- *Dynamic and flexible:* The Liberty profile server runtime loads only what your application needs and constructs the run time in response to configuration changes.
- *Extensible:* The Liberty system programming interfaces (SPIs) provide support for user and product extensions that can use the SPIs to extend the run time. An example of this is the additional features that CICS TS provides in the CICS Liberty JVM server (see 2.4, “Liberty in the CICS Transaction Server” on page 18).

2.2.1 Liberty and the CICS Transaction Server for z/OS Value Unit Edition

In addition to the technical possibilities that Liberty within CICS makes available, applications that are written and run within CICS Liberty qualify as new Java workloads and are eligible to run within a CICS Transaction Server for z/OS Value Unit Edition region on a zNALC LPAR.

New Java-based features can be developed within Liberty and can be self-contained. For example, they can interact with IBM MQ Value Unit Edition or IBM DB2 Value Unit Edition also running on the zNALC LPAR. Another possibility is to use the Java class library for CICS (JCICS) to link to an existing application running in a standard CICS Transaction Server for z/OS region on a standard LPAR.

2.3 Strengths

Liberty offers great advantages when used as both a development run time and production run time within CICS TS. Liberty is both lightweight and capable, particularly when considering the ability of third-party products to extend and enhance the available features. Liberty offers great advantages when used as both a development and production runtime environment in CICS TS.

2.3.1 Simple configuration

The server configuration, from the server administrator's perspective, is only a single `server.xml` file that contains all necessary information. The configuration file for IBM WebSphere Application Server Liberty Profile for z/OS V8.5.5 (the version included in CICS TS V5) has many optional parameters that you can use for specific scenarios. They are listed in "Liberty features" in the CICS Transaction Server section of the IBM Knowledge Center:

http://ibm.co/1zgtbPv

2.3.2 Runtime composition with features and services

The composable nature of Liberty is based on the concept of features. A *feature* is a function. Features can overlap, and they can include other features.

CICS Liberty JVM server consists of a JVM server that hosts the Liberty kernel and several optional features. The feature code and most of the kernel code run as OSGi bundles within an OSGi framework. Features provide the programming models and services required by applications. You can choose which optional features should be enabled according to your application requirements.

Note: Only one CICS Liberty JVM server can run per CICS region with security enabled. If multiple Liberty profile servers need to run within a single CICS region, security must be disabled within the JVM profile for the JVM server. By default, security is enabled, so if the required flag is not set within the JVM profile, only one CICS Liberty JVM server will start.

2.3.3 Developer focus

With Liberty, you can do rapid development and deployment to meet with modern development trends. Liberty offers the following advantages for developers:

- Fast and no-cost download for developer's workstation

  Liberty profile server is fast and no cost for developer workstation use. It can be downloaded and installed from Eclipse.org or WASdev.net.

- Rapid development and deployment

  You deploy an application in Liberty profile server either by dropping the application into server's `drop-ins` directory or by adding an application entry to the server configuration (`server.xml`) file.

  In addition to this, within CICS Liberty JVM server, you can package an application, for example an enterprise bundle archive (EBA) in a CICS TS bundle, and deploy this bundle within CICS TS. This is the suggested deployment method for applications in CICS Liberty JVM server.

- Easy extensibility for custom features and third-party components

  Liberty supports direct extension of the runtime environment using *product extensions*. A product extension allows custom content to be added to a Liberty installation in a way that avoids conflicts with the base content of the product and with other product extensions.

2.4 Liberty in the CICS Transaction Server

As previously mentioned, CICS Transaction Server (TS) runs Liberty within a CICS TS Java virtual machine (JVM) server. Figure 2-1 shows the basic architecture of how a Liberty profile server is hosted within a CICS TS region.

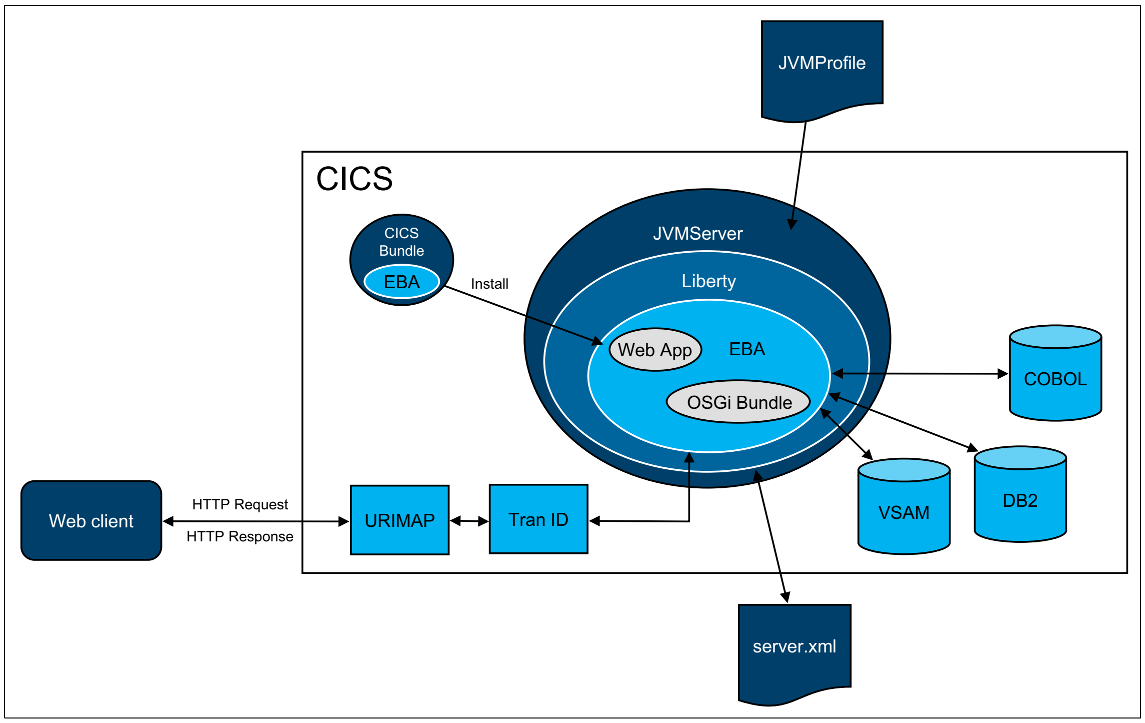


Figure 2-1 Architecture of Liberty in a CICS TS JVM server

As Figure 2-1 illustrates, the Liberty profile server is hosted in a CICS TS JVM server that is defined in a JVM profile. This example shows how a CICS TS bundle containing an EBA is placed within CICS TS and then installed in the Liberty environment. This EBA, which contains a web application, is then able to access CICS TS resources, for example to IBM DB2 or VSAM data sets or to other CICS TS COBOL applications.

This example demonstrates using a URIMAP as an entry point into CICS Liberty JVM server from a web client, which allows context switching of transaction ID or user ID. Liberty listens on a port and handles HTTP traffic, with the ability to define transport security (see 2.5.2, “Security overview” on page 21). If context switching is not required, a web client can simply connect directly to Liberty.

Figure 2-1 on page 18 also shows that the `server.xml` file that defines the configuration of the CICS Liberty JVM server is also available, although this can be automatically generated by CICS TS. Security aspects of this architecture are described later in this chapter.

CICS TS TS V5.2 supports a subset of the total features of the full Liberty profile. These supported features include, but are not limited to the following:

- JavaServer Faces (JSF) 2.0
- JavaServer Pages (JSP) 2.2
- Java Servlet 3.0
- Java API for RESTful Web Services (JAX-RS) 1.1
- Java API for XML Web Services (JAX-WS) 2.2
- JavaScript Object Notation (JSON4J) 1.0
- Secure Socket Layer (SSL) 1.0
- Web Application Bundles (WAB) 1.0
- Java Database Connectivity (JDBC) 4.0

You can find a complete list of supported Liberty features in the CICS section of the IBM Knowledge Center:

`http://ibm.co/1yD5Ed6`

CICS Liberty JVM server includes all features of the full Liberty profile (in CICS TS V5.2, this is the WebSphere Application Server Liberty profile for z/OS V8.5.5.1). However, the features that appear in the IBM Knowledge Center are a subset that are currently supported. You may choose to include other features in your applications, but this is not supported.

In addition to the Liberty features supported in CICS TS, the following are also provided:

- CICS TS Core (`cicsts:core-1.0`)

  Provides core CICS TS features and Java Transaction API (JTA) 1.0
- CICS TS JDBC (`cicsts:jdbc-1.0`)

  Provides support for applications to access a local CICS TS DB2 database using JDBC.
- CICS Liberty JVM server security (`cicsts:security-1.0`)

  Provides integration of Liberty security with CICS TS security, including propagation of thread identity. This feature includes the *zosSecurity-1.0* feature.

2.4.1 Integration with CICS TS Transaction Server for z/OS

The two subsequent chapters in this part will outline the advantages of web applications running in a Liberty JVM server in a CICS TS environment. The focus is to allow new applications, whether they be new presentation layers to interact with current CICS TS applications, or applications that will use resources managed by CICS TS, to benefit from collocation with the CICS TS environment. Both use cases focus on the ease of creating new applications that can integrate with CICS TS.

The two main ways to interact with CICS TS are to call an existing application, for example a CICS COBOL application running on an existing non-zNALC LPAR, or new applications, to take advantage of the JCICS API to call CICS TS functions. There are advantages to each approach and the choice depends mainly on what you are trying to achieve. If the main objective is to modernize your application presentation layer, for example implementing a Java API for RESTful Web Services (JAX-RS) interface allowing interaction with mobile applications, the appropriate choice maybe to link to existing applications. Similarly, if a new application is being developed from scratch, you can implement all functions from Java within Liberty. This new application, as well as calling the JCICS API directly to access resources could also call existing applications hosted on another LPAR.

Before these options are explored further in the rest of this part, the rest of this chapter will outline what security options are available within CICS Liberty JVM server and some security considerations that are important to highlight.

Note: While it is possible for an application hosted in CICS Liberty to link to a CICS COBOL, or other type of application in CICS, the ability to link to web applications running in a Liberty JVM server is not supported in CICS TS V5.2. If this is required, a CICS COBOL application could make an HTTP call to web applications running in a Liberty JVM server.

2.5 Security

This section will provide an introduction to the security features that are available for CICS Liberty JVM server applications.

The security features mentioned below may, and probably will, form part of a wider security infrastructure. It is likely that further levels of security will be required for external users to access applications hosted in CICS Liberty JVM server, for example integrating with IBM MobileFirst Platform Foundation. This chapter covers only the security features specific to CICS Liberty JVM server applications. Also see 5.5, “IBM MobileFirst Platform Foundation and CICS TS” on page 54.

2.5.1 Introduction to security with Liberty in CICS TS

CICS Liberty JVM server includes *appSecurity-2.0*, which enables security for web applications when the *servlet-3.0* feature is present. This complements SSL support (through the ssl-1.0 feature), which enables SSL (including TLS) connections using HTTPS (see 2.5.2, “Security overview” on page 21).

For CICS TS specific security options, CICS TS has an additional feature, *cicsts:security-1.0*, which has integrated *zosSecurity-1.0* features with CICS TS security options, for example propagation of user identity. To use the *cicsts:security-1.0* feature, the angel process that is available in CICS TS TS V5.2 must be running. This is used for accessing z/OS authorized services and can use SAF security frameworks. The angel process is the suggested authentication and authorization method for CICS Liberty JVM server applications. See 2.5.3, “The Liberty server angel process” on page 22 for more details on the angel process.

Furthermore, CICS Transaction Server for z/OS V5.2 includes improved performance through using the Liberty authentication cache. When a user is authenticated a new *Subject* object is created storing all authorization information, including roles. This Subject object is stored in the Liberty authentication cache, with a configurable cache expiry time, preventing multiple authentication and authorization requests for the same user.

2.5.2 Security overview

Before proceeding to read about the security options that are available in Liberty in CICS TS, be sure to review the following key terms that are essential for understanding security:

- **Transport (communication)**

  Transport, or communication, refers to the mechanism that is used for data to travel between two places, for example from a client (for example a web browser on a computer) to a server (for example an application in Liberty running within CICS TS). When referred to in this chapter we are concerned with how to secure the data that is sent between two sources.

  Communications in CICS Liberty JVM server are secured with the Secure Sockets Layer (SSL) protocol. The SSL protocol provides transport layer security including authenticity, data signing, and data encryption to ensure a secure connection between a client and server. SSL in CICS Liberty JVM server includes TLS v1.2 support required for some security standards and the protocol used can be configured in `server.xml`, within the `<ssl>` element using the `sslProtocol` attribute.

  You can configure a CICS Liberty JVM server JVM server to use SSL for data encryption, and optionally authenticate with the server using a client certificate. Client certificates can be stored in a Java keystore or in a SAF key ring.

- **Authentication**

  Authentication confirms that an entity (for example a user) that is attempting to access a resource (for example an application hosted in Liberty) is a valid entity. Typically, this entity will provide a username and password when attempting to gain access. This username and password, or possibly a client certificate, is used to authenticate the entity.

  CICS Liberty JVM server includes many different options to aid in authentication. For an introduction on Liberty authentication visit this website:

  http://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/cwlp_authentication.html

  CICS Liberty JVM server supports several of the authentication options described on that web page, with some additions. The basic Liberty authentication features supported in CICS Liberty JVM server are:

  – SSL client authentication
  – Form logon
  – Lightweight Third-Party Authentication
  – Custom user registry
  – Trust Association Interceptor

  In addition, the angel process described in 2.5.3, "The Liberty server angel process" on page 22 allows authentication using z/OS security services (SAF) is required for applications that interact with other CICS TS processes and are contained within CICS TS bundles.

- **Authorization**

  Authorization determines whether a given entity has been granted the correct privileges in order to access a resource. This can be used, for example, in protecting certain areas of a website that may only be available to certain authorized users.

  For an introduction on Liberty authorization, visit this website:

  http://www.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/cwlp_authorization.html

In addition to the <application-bnd> configuration options available in Liberty, CICS Liberty JVM server includes the following addition authorization options:

– Roles (defined as SAF EJBROLE) - described in 2.5.4, “SAF roles” on page 23
– The angel process authorization - described in 2.5.3, “The Liberty server angel process” on page 22
– ThreadIdentityService (can push Subject credential onto CICS TS task)

- **Role**

  Typically, authorization is aided by the use of roles. An entity can be assigned one or more roles and then a resource can be authorized to be used by a role. If the entity is a member of a role, and a role is authorized to access a resource, access is granted to the entity.

- **Subject**

  A subject is a representation of a given entity, for example, as mentioned before, a user. When a user is authenticated, their authorization information (for example Role membership) is retrieved and stored in a Subject object within Liberty authentication cache. Any subsequent authentications will result in this Subject object being retrieved from the cache, improving performance. The Liberty authentication cache has a configurable expiry time for the Subject object, with a default of ten minutes.

2.5.3 The Liberty server angel process

As described in 2.5.1, “Introduction to security with Liberty in CICS TS” on page 20, CICS Transaction Server for z/OS V5.2 added support for the angel authentication and authorization process to CICS Liberty JVM server. It is now the default for authentication and authorization in CICS Liberty JVM server.

The angel process is lightweight and does very little CPU-consuming work after establishing control blocks. Only one angel process is required per z/OS operating system image (logical partition, LPAR). This process has no configuration files and uses no TCP/IP ports.

By deafult, the angel process uses the System Authorization Facility (SAF) user registry for all authentication requests if the CICS Liberty JVM server security feature is included. This allows a Liberty-based application hosted in CICS TS and other CICS TS processes that are linked to from CICS Liberty to use the same user identity.

Figure 2-2 shows how authentication and authorization requests are routed through the angel process if the CICS TS security feature is present.

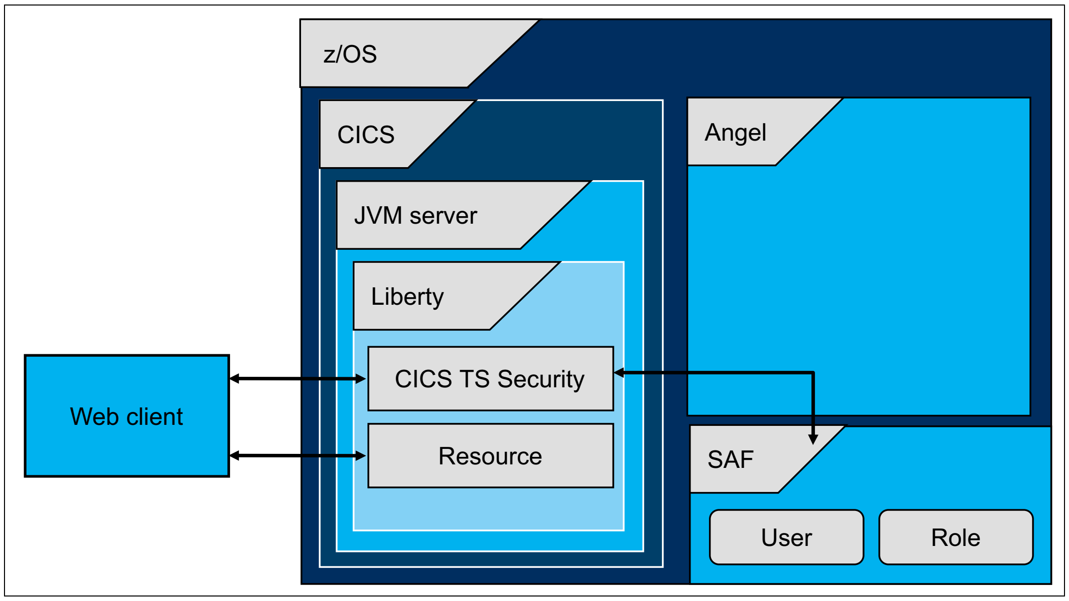


Figure 2-2 Process diagram for authentication and authorization with the angel process

Figure 2-2 shows a web client connecting to a Liberty profile server running within CICS and requesting access to a resource. The CICS TS security feature (cicsts:secuity-1.0) has been configured in the Liberty profile server. When the web client requests access to the resource, the request is routed through the angel process onto SAF, where the user credentials are authenticated and checked against the requested resource. The resource access could be controlled by using roles. The user must be part of the role to be granted access. After the user is authenticated and authorized, the response to the authorization request is passed back to Liberty. If that is successful, the resource is returned to the web client.

For more information about how to configure CICS Liberty JVM server with the angel process, see “The Liberty server angel process” in the IBM Knowledge Center:

http://ibm.co/12nSggY

2.5.4 SAF roles

When you are using the CICS Liberty JVM server security feature, you can include a configuration element, `<safAuthorization>`, in the `server.xml` file to enable the use of SAF roles (EJBROLE). If this is present any roles defined in `server.xml` are ignored and role membership is defined and granted using SAF roles. An EJBROLE can be defined using SAF and then membership of that role is granted to users defined in the SAF registry, allowing access and permissions to SAF authorized resources for example CICS TS bundles containing CICS Liberty JVM server applications.

For more information about how to configure roles within CICS Liberty JVM server, see “JEE application role security” in the IBM Knowledge Center:

http://ibm.co/1HXlPGy

Note: The chapters that follow describe the main scenarios for creating new applications or migrating existing ones to CICS Liberty JVM server and explain the advantages and possibilities.

3

Using CICS Liberty JVM servers to develop application interfaces

This chapter describes how to implement the modernization of presentation in IBM CICS Transaction Server (TS) with CICS Liberty Java virtual machine (JVM) server. We include scenarios to develop CICS Liberty JVM server applications to gain the benefit of IBM CICS Transaction Server for z/OS Value Unit Edition and the features that the Liberty JVM server provides to run the presentation logic. We also introduce a toolkit to migrate an existing Java application from other platforms to CICS Liberty JVM server.

This chapter covers the following topics:

3.1 CICS Liberty JVM server scenarios

Before CICS Liberty JVM server, CICS TS supported the 3270 screen and the web as the *presentation* interfaces, with CICS TS Basic Mapping Support (BMS), CICS TS Web Support (CWS), or CICS TS Dynamic Scripting Feature Pack.

The 3270 screen is a traditional mainframe interface that is not as modern as a web or mobile interface. There are many requirements from IBM clients who use CICS TS to replace the 3270 screen with web browser and RESTful clients.

CICS TS Web Support can work with web applications, but the application programmers need to use CICS TS web APIs to analyze the HTTP data and to assemble the HTTP response. Web applications are typically no longer developed in this way.

The Dynamic Scripting feature pack was originally included with CICS Transaction Server for z/OS Version 4 Release 1. It supports the use of PHP and Groovy to develop situational applications. In CICS Transaction Server for z/OS Version 5 Release 1, the Dynamic Scripting feature pack runs in CICS Liberty JVM server and it only supports PHP. We explain more in 3.2, “CICS Liberty JVM server features for the presentation layer” on page 28.

There are other third-party presentation technologies used to connect to CICS TS. Some CICS TS customers are eager to modernize these client connectors with web front end (servlets).

CICS Liberty JVM server is the preferred way to develop web applications in CICS TS or to handle RESTful request. All Java workloads deployed using CICS Liberty are candidates for approval for IBM CICS Transaction Server for z/OS Value Unit Edition.

There are two typical scenarios for customers to use CICS Liberty JVM server:

- Development of a Java solution for a new business requirement or replacement
- Replacement of an older presentation layer with a modern presentation layer

3.1.1 Scenario one

The first scenario is that a customer wants to develop a pure Java solution either for a new business requirement or as a replacement of existing solution. The Java solution includes presentation logic, business logic and data access logic. In this case, everything can be deployed and run in a zNALC-enabled LPAR.

In Figure 3-1 on page 27, we show a general architecture for this scenario. We have a CICS TS region running in a zNALC-enabled LPAR. In the region, we set up a JVM server for CICS Liberty JVM server. We develop an application with presentation and application interfaces, business logic, and data access. In the presentation layer, you can use the features that CICS Liberty JVM server provides. For more information, see 3.2, “CICS Liberty JVM server features for the presentation layer” on page 28.

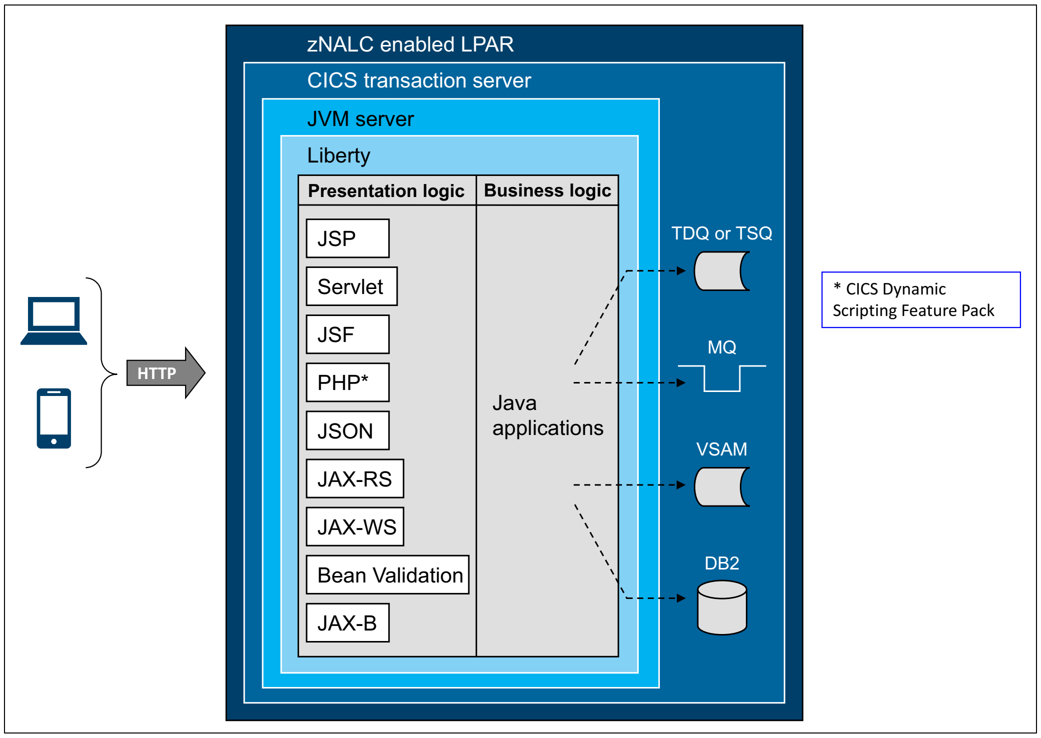


Figure 3-1 Presentation, business logic, and data access in CICS Liberty JVM server

To develop and deploy a CICS TS application, the CICS TS Explorer SDK is a preferred tool. It can be downloaded at no charge from the IBM website and installed in an Eclipse integrated development environment (IDE). The development process is similar to the process used to develop a Liberty project for a distributed platform. To access the data, CICS TS provides the JCICS API support.

After the development work is done, the Java web application, either in the form of WAR files or an EBA file, can be deployed as one or more CICS TS bundles in a z/OS file system (zFS). If there are common Java classes to be shared by multiple versions of different Java web applications, you can deploy these classes as common OSGi bundles to zFS directories and refer to these directories from a `bundleRepository` element of the `server.xml` file of the Java web applications.

For more information about how to develop Java applications in CICS Liberty JVM server, see Chapter 4, “Porting JEE applications to a CICS Liberty JVM server” on page 33.

3.1.2 Scenario two

Another scenario is that there is a need to replace older presentation layers (for example, BMS) with a more modern presentation layer, such as a web or RESTful interface. In this case, existing business logic and data access can be reused. You can develop and deploy the new presentation layer in CICS Liberty JVM server that runs in a zNALC-enabled LPAR. The presentation layer can then link to the existing business logic, which is still in a standard z/OS LPAR. Figure 3-2 on page 28 shows the architecture for this scenario.

.

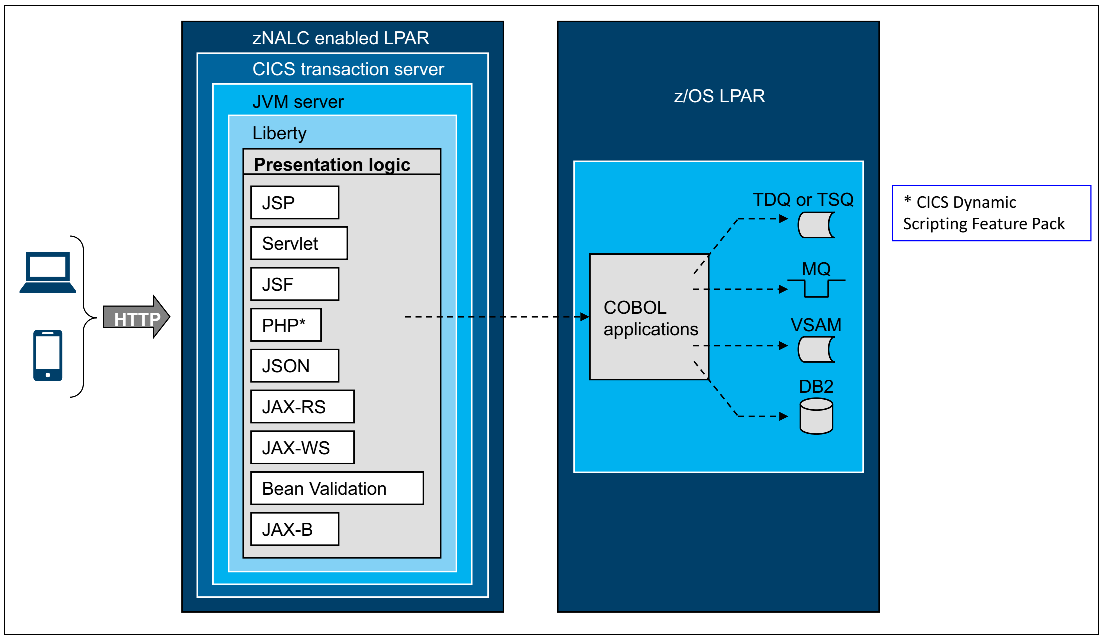


Figure 3-2 Presentation in CICS Liberty JVM server and link to existing business logic

The development and deployment in this scenario is the same as the previous scenario. The difference is that we focused on only the presentation layer. To communicate with an existing business logic in another CICS region, distributed program link (DPL) is a preferred way. *Channels and containers* and *COMMAREA* are both supported to carry the data.

Many existing CICS TS applications are centered on the use of structured data records, which are typically stored as sequential files in the Virtual Storage Access Method (VSAM) data set or as DB2 tables. To communicate with these existing applications, it is frequently desirable to reuse the copybooks, which describe these existing record structures. To simplify the interactions from the Java application with the structured data required by the existing CICS TS applications, IBM provides utilities such as JZOS and J2C to make the task easier. For more information about how to use JZOS and J2C, see Chapters 8 in the IBM Redbooks publication titled *IBM CICS and the JVM server: Developing and Deploying Java Applications*, SG24-8038:

http://www.redbooks.ibm.com/abstracts/sg248038.html

3.2 CICS Liberty JVM server features for the presentation layer

CICS TS uses some of the features in the WebSphere Application Server Liberty Profile to run the following in a JVM server:

Web applications	JavaServer Pages (JSP), JavaServer Faces (JSF), and so on
Web services	Java API for XML Web Services (JAX-WS) and Java Architecture for XML Binding (JAXB)
RESTful services	JavaScript Object Notation (JSON) and Java API for RESTful Web Services (JAX-RS)

For a list of features that CICS TS V5.2 supports, see "Liberty features" in the IBM Knowledge Center:

`http://ibm.co/1FPFtlA`

In this section, we briefly introduce the Liberty features that CICS support for the presentation layer. As mentioned, these features are from the WebSphere Application Server Liberty profile. For more information about how to use these features, see the IBM Redbooks publication titled *WebSphere Application Server Liberty Profile Guide for Developers*, SG24-8076:

`http://www.redbooks.ibm.com/abstracts/sg248076.html?Open`

3.2.1 JavaServer Pages 2.2

JavaServer Pages (JSP) are a widely used technology to easily create dynamic web pages based on HTML, XML, or other document types. JavaServer Pages enable the separation of the Hypertext Markup Language (HTML) code from the business logic in web pages so that HTML programmers and Java programmers can more easily collaborate in creating and maintaining pages.

CICS Liberty JVM server supports the JavaServer Pages 2.2 specification. For more information about JSP and how to develop JSP in the Liberty profile, see "Developing JSP files" in the IBM Knowledge Center:

`http://ibm.co/1w2NNgN`

3.2.2 JavaServer Faces 2.0

JavaServer Faces (JSF) simplify the development of user interfaces for web applications by providing the following features:

- Templates to define the layout
- Composite components that turn a page into a JSF UI component
- Custom logic tags
- Expression functions and validation
- Component libraries
- XHTML page development

For more information, see "JavaServer Faces" in the IBM Knowledge Center:

`http://ibm.co/1zliJWI`

3.2.3 Java Servlet 3.0

Java servlets are a widely used technology for building dynamic content for web-based applications. The servlet Java classes are used to extend the capabilities of a server, commonly a web server.

CICS Liberty JVM server provides support for HTTP servlets written to the Java Servlet 3.0 specification. For more information about developing servlets, see the following website:

`http://ibm.co/1ygLVBd`

3.2.4 JavaScript Object Notation 1.0

The JavaScript Object Notation (JSON4J) library is an implementation of a set of JavaScript Object Notation (JSON) handling classes for use within Java environments.

The JSON4J library provides the following functions:

- A simple Java model for constructing and manipulating data to be rendered as the JSON implementation.
- A fast transformation of XML for JSON conversion for situations where you want conversion from an XML reply from a web service to a JSON structure for easy use in Asynchronous JavaScript and XML (Ajax) applications.
- A JSON string and stream parser that can generate the corresponding JSONObject, which represents that JSON structure in Java. You can then change that JSONObject and serialize the changes back to the JSON implementation.

CICS Liberty JVM server provides access to the JSON4J library, which includes a set of JSON handling classes for Java environments. For more information, see “JavaScript Object Notation (JSON4J)” in the IBM Knowledge Center:

http://ibm.co/1w2Ol6o

JSON4J can be used to connect mobile devices to CICS TS. For more information, see Chapter 5, “Connecting mobile devices to CICS Transaction Server” on page 47.

3.2.5 Java API for RESTful Web Services

Java API for RESTful Web Services (JAX-RS) is a technology to develop services that follow Representational State Transfer (REST) principles. RESTful services are based on manipulating resources. Resources can contain static or dynamically updated data. By identifying the resources in your application, you can make the service more useful and easier to develop.

CICS Liberty JVM server provides support for the Java API for RESTful Web Services on the Liberty profile. For more information about JAX-RS, see “Developing web services - RESTful services” in the IBM Knowledge Center:

http://ibm.co/15PABQL

JAX-RS can also be used to connect the mobile device to CICS TS. For more information, see Chapter 5, “Connecting mobile devices to CICS Transaction Server” on page 47.

3.2.6 Java API for XML Web Services 2.2

Java API for XML-based Web Services (JAX-WS) is the next-generation web services programming model. Using JAX-WS, development of web services and clients is simplified, with more platform independence for Java applications, by the use of dynamic proxies and Java annotations.

CICS Liberty JVM server supports SOAP web services that are based on JAX-WS 2.2. For more information, see “JAX-WS” in the IBM Knowledge Center:

http://ibm.co/1vNzlaD

JAX-WS can also be used to connect the mobile t CICS TS. For more information, see Chapter 5, “Connecting mobile devices to CICS Transaction Server” on page 47.

3.2.7 Java Architecture for XML Binding 2.2

Java Architecture for XML Binding (JAXB) is a Java technology that provides an easy and convenient way to map Java classes and XML schema for simplified development of web services. JAXB leverages the flexibility of platform-neutral XML data in Java applications to bind XML schema to Java applications without requiring extensive knowledge of XML programming. JAXB provides the XJC schema compiler tool and the schemagen schema generator tool to transform between XML schema and Java classes.

CICS TS provides JAXB support to map between Java classes and XML representations. For more information, see "JAXB" see in the IBM Knowledge Center:

http://ibm.co/1vloSOX

JAXB can also be used to connect the mobile device to CICS TS. For more information, see Chapter 5, "Connecting mobile devices to CICS Transaction Server" on page 47.

3.2.8 Bean Validation 1.0

It is always quite important to validate input received from the user to maintain data integrity in application logic. For example, it is necessary to validate the input of an email address before sending email. Bean Validation is a new validation model that is available as part of the Java Enterprise Edition 6 platform. The model is supported by constraints in the form of annotations placed on a field, method, or class of a JavaBeans component.

The Bean Validation API is introduced as a standard mechanism to validate enterprise JavaBeans in all layers of an application, including presentation, business, and data access. Before the Bean Validation specification, the JavaBeans were validated in each layer. To prevent the reimplementation of validations at each layer, developers bundled validations directly into their classes or copied validation code, which was often cluttered. Having one implementation that is common to all layers of the application simplifies the developer's work and saves time.

With Bean Validation, CICS Liberty JVM server provides validations for JavaBeans at each layer of an application. For details, see "Bean Validation" in the IBM Knowledge Center:

http://ibm.co/1zlxfxU

3.2.9 PHP support by Dynamic Scripting Feature Pack

PHP support is a feature provided by CICS TS Transaction Server Feature Pack for Dynamic Scripting V2.0, not the WebSphere Application Server Liberty profile. The feature pack provides an agile web application platform for developing and running modern web applications. You can use the Feature Pack for Dynamic Scripting V2.0 to create and run PHP applications that meet your specific needs or the needs of your clients. For more information, see "CICS Transaction Server for z/OS Feature Pack for Dynamic Scripting V2.0" in the IBM Knowledge Center:

http://ibm.co/1tOHAO2

3.3 Migrate existing Java presentation logic to CICS Liberty JVM server

The WebSphere Application Server V8.5.5 Liberty profile provides a simple tool to analyze what Java API packages are supported according to feature. For example, the following command produces an XML file that shows all of the API packages for each feature, which can be cross-checked against the Liberty features supported in CICS TS:

```
$USSHOME/wlp/bin/featureManager featureList
--encoding=ibm037/tmp/wlp855_featureList.xml
```

Migrating a Java application from one platform to another is easier than the migrating applications developed in other languages. If you have existing Java presentation logic outside of CICS TS and want to migrate the logic to CICS Liberty JVM server, the IBM WebSphere Application Server Migration Toolkit can make the migration easier.

Note: Make sure that the features used in existing Java applications are supported by CICS Liberty.

The WebSphere Application Server Migration Toolkit is a suite of tools and collections of knowledge that enable your organization to quickly and cost-effectively migrate to WebSphere Application Server V7.0 through V8.5.5, whether from a previous version of WebSphere Application Server or competitive application servers, including Apache Tomcat Server, JBoss Application Server, Oracle Application Server, and Oracle WebLogic Server.

For more information, see the “IBM WebSphere Application Server Migration Toolkit” page on IBM developerWorks®:

http://www.ibm.com/developerworks/websphere/downloads/migtoolkit/

The Migration Toolkit can help you complete part of the migration, but not all. You still need to tailor your application to make it work within CICS. For example, the data access in mainframe is quite different from that in a distributed platform. You might need to move some recoverable application data to VSAM or DB2, and then use the JCICS API or JDBC to access to the data. For more information, see Chapter 4, “Porting JEE applications to a CICS Liberty JVM server” on page 33.

4

Porting JEE applications to a CICS Liberty JVM server

This chapter describes considerations for porting existing Oracle Java Enterprise Edition (Java EE) and other existing Java applications to an IBM CICS Liberty profile Java virtual machine (JVM) server. It also explains considerations for developing Java applications that use features that are unique to a CICS Liberty JVM server and other Liberty features.

It includes the following topics:

4.1 Porting a Java application to a CICS Liberty JVM server

The features in a release of CICS Liberty JVM server are a subset of the features supported by the corresponding release of the Liberty profile that is provided in IBM WebSphere Application Server for z/OS. Existing JEE and other Java applications can be ported or migrated to a CICS Liberty JVM server if the applications use only features that are supported by the currently installed release level of the CICS Liberty JVM server.

The supported server features for a release are determined by which features have been verified by the CICS Transaction Server (CICS TS) development organization. For a list of the currently supported features, see the “Liberty features” web page:

http://ibm.co/1yD5Ed6

Other enhancements may be available and should work in a CICS Liberty JVM server, with certain caveats or restrictions. For information about these enhancements, check the CICSdev Community blog on IBM developerWorks:

https://www.ibm.com/developerworks/community/blogs/cicsdev

One example of such an enhancement is the addition of support for IBM WebSphere MQ messaging software, which is described in the article titled “Using the WebSphere MQ classes for Java with a CICS JVM server:”

Note: To determine which features are available in a particular instance of CICS Liberty JVM server, locate the product service signature string in the Liberty startup messages, such as this example:

```
product = CICS Transaction Server for z/OS 5.2.0, CICS Liberty JVM server
NOTUSAGE, WebSphere Application Server 8.5.5.1, WAS FOR Z/OS 8.5.5.1
(wlp-1.0.4.cl50120140502-1451)
```

In this example, the features included with release 8.5.5.1 of the WebSphere Application Server for z/OS Liberty profile are available.

4.1.1 Which Java applications should be migrated to CICS TS

Not every Java application that meets the previously described criteria should be ported to CICS Liberty JVM server. Consider porting Java applications only when there is a logical reason or a need for the application to run in a CICS TS JVM server.

For example, a Java application should be migrated to CICS Liberty JVM server in these situations:

- If you have an Java application that interacts with a CICS TS application using CICS TS Transaction Gateway, CICS TS MQ Bridge, or a JEE Connector (J2C) connection factory. Consider moving this Java application to Liberty JVM server to take advantage of being colocated with the CICS TS application that is being accessed by the user interface.
- If you want to start using some of the features to Liberty JVM server in CICS. For example, the JDBC features that allow dynamic binding of a data source at run time or the feature that coordinates the commits or rolls back updates made to remote data sources (JDBC Type 4) in the same logical unit of work as other CICS TS managed resource.
- If you want to take advantage of using CICS bundles for your applications. Using bundles means that application artifacts can be packaged with the corresponding CICS TS resource definitions (that is, definitions for TRANSACTIONs, PROGRAMs, URIMAPs, and

so on) so that both the application and its resource definitions can be managed or deployed as a single administrative entity.

Because CICS Liberty JVM server runs in a CICS TS JVM server, all of the inherent advantages provided to Java applications by the CICS JVM server are available to CICS Liberty JVM server applications. For example:

- Rather than having multiple JVMs running a CICS TS task, there is a single JVM in a JVM server that can spawn up to 256 threads, with each thread executing a CICS TS task. This provides vertical scaling as workload increases. A CICS Liberty JVM server can also be configured with only the features required by an application to execute. Eliminating the features not required in the server avoids loading unnecessary functions and reduces server startup time and storage use.
- Multiple JVM severs can coexist is a single CICS TS region, with each JVM server configured independently and running different applications. Each JVM server (CICS Liberty and non-Liberty), running different applications, can horizontally scale application workload within a CICS region. Currently, there can be only one CICS Liberty JVM server in a CICS region that interacts with a Liberty angel process, using IBM z/OS security features. For horizontal scaling of CICS Liberty applications that required z/OS security, the CICS Liberty JVM server needs to be replicated in another CICS region.

For more information about running Java in CICS TS JVM servers and the Open Service Gateway initiative (OSGi), download the white paper titled “Running Java workloads with JVM servers and OSGi:”

http://ibm.co/12gJqBL

4.1.2 Using the OSGi framework

Another consideration for consolidating applications in CICS Liberty JVM server is to use the OSGi framework for deploying and administering Java applications. The framework restructures the components of an application as individual bundles of components or packages that are loosely coupled but constitute an application when combined. This contrasts with packaging solutions where all components are packaged and administered in a single Java archive (JAR) file that must be included in the class path. This separation of the application into bundles enables independent deployment and management of the different bundles that compose an application and, potentially, reuse of selected bundles by other applications.

This flexibility has key benefits:

- Different versions or levels of the same package can coexist in CICS Liberty JVM server concurrently.
- Because the Java class path is not used to load the Java classes, changes can be implemented without the need to stop and restart the JVM.

There are various development tools available (such as CICS TS Explorer or Eclipse integrated development environment [IDE] tools, such as Luna or Kepler) to aid in refactoring existing Java artifacts, such as JARs, into OSGi bundles for deployment to bundle repositories and packaging applications into CICS TS bundles for deployment to a CICS Liberty JVM server run time.

For more information about OSGi packaging in CICS Liberty JVM server, see “The OSGi Service Platform” in the IBM Knowledge Center:

http://ibm.co/1CGxIQT

For more information about Liberty application development in general and to download Eclipse and Liberty development plug-ins, go to the following web pages:

https://developer.ibm.com/wasdev

or

http://www.eclipse.org/downloads/packages/

To download the CICS Explorer system management tool, start at this web page:

http://www.ibm.com/cics/explorer

Note: Both Eclipse tools and CICS Explorer work with common source code management products, such as IBM Rational® Team Concert.

4.2 Developing new application using JCICS classes

CICS TS provides Java interfaces to most of the same standard CICS TS application programmer interfaces (APIs) that are in traditional CICS TS application programs. New Java applications can be developed or existing Java applications can be modified to access almost the full set of CICS TS APIs. Use of these APIs gives Java applications running in a CICS TS JVM server full access to VSAM files, the ability to link to COBOL and other CICS application programs passing common data areas and containers, the ability to access transient data and temporary storage queues, and so on.

Example 4-1 and Example 4-2 are Java code that shows some the Java class library for CICS (JCICS) classes and methods that can be used in a CICS Liberty JVM server application.

Example 4-1 JCICS example of accessing a key sequenced VSAM file

```
// Instantiate an instance of a JCICS record holder
com.ibm.cics.server.RecordHolder record = new
   com.ibm.cics.server.RecordHolder();
// Instantiate an key sequence VSAM file
com.ibm.cics.server.KSDS file = new com.ibm.cics.server.KSDS();
// Set the file name
file.setName("FILEA");
// Retrieve and delete a record in a Try/Catch block
try {
   file.read(key.getBytes(),record);
   file.delete(key.getBytes());
catch (Exception e) {
```

Example 4-2 JCICS examples of using channels and containers in linking to a CICS TS program

```
// Instantiate an instances of a JCICS channel and a JCICS container
Channel channel = task.createChannel("MINICICS-Channel");
Container requestContainer = channel.createContainer("MINICICS");
// Copy contents of the CICS commarea area to the JCICS request container
requestContainer.put(commarea.getByteBuffer());
// Invoke program and pass the channel with the request container
progName.link(channel);
// Retrieve the response container from the channel
Container responseContainer = channel.getContainer("MINICICS");
// Copy contents of the response container to the CICS commarea area
commarea = new com.ibm.ats.odm.COMMAREA(responseContainer.get());
```

CICS Explorer and Eclipse development tools can also be used to develop Java applications that use JCICS. Accessing CICS TS resources by using JCICS classes should be more understandable to Java programmers than the EXEC CICS equivalents used in other languages.

Note: Some JCICS classes and methods should not be used in CICS Liberty JVM server. For example, any API that deals with a terminal (that is, methods such **`clear`**, **`converse`**, **`erase`**, and so on) would not be used because these require a principle facility or terminal and there would be no terminal associated with a task running in CICS Liberty JVM server.

4.2.1 Java access to records and their fields

Eventually, most applications running in CICS TS need to work with a record and its fields when accessing a VSAM file, a DB2 database row, and so on. Most Java development on other platforms work with stream-oriented data rather than with record-oriented data as typical applications in CICS TS. Because of this, there are options for generating a Java class that represents the layout of records and the corresponding `set` and `get` methods for accessing the individual fields in the record. Example 4-3 shows use of `set` and `get` methods.

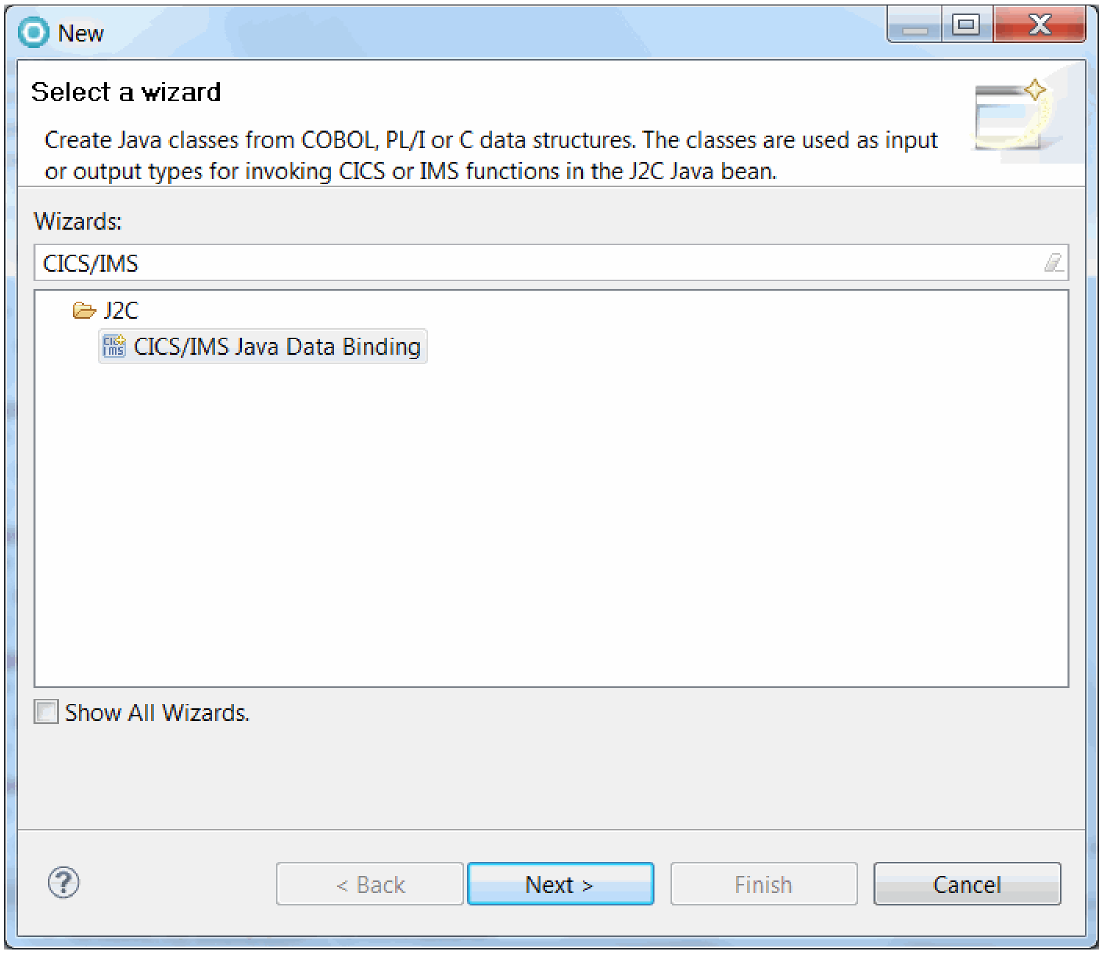
Example 4-3 Example of the using of get and set methods with a COMMAREA class

```
commarea.setName(request.getParameter("custName"));
   commarea.setEffectdate(request.getParameter("effectDate"));
   commarea.setAmount(Long.parseLong(request.getParameter("amount")));
   commarea.setAge(Long.parseLong(request.getParameter("age")));

   // Invoke a CICS COBOL program passing a common area
   commarea = MiniCICS.invoke(commarea);

   request.setAttribute("custName",commarea.getName());
   request.setAttribute("age", commarea.getAge());
   request.setAttribute("amount", commarea.getAmount());
   request.setAttribute("effectDate", commarea.getEffectdate());
```

The subsections that follow describe some of the options for generating Java classes from record-oriented layouts.

Rational Application Developer J2C wizard

In IBM Rational Application Developer, the Java EE Connectors feature has a J2C component that includes a wizard that generates a Java data bean with `get` and `set` methods. These methods can be used to access individual fields within a record. Supported languages include COBOL, C, and PL/I (see Figure 4-1).


New

Select a wizard

Create Java classes from COBOL, PL/I or C data structures. The classes are used as input or output types for invoking CICS or IMS functions in the J2C Java bean.

Wizards:

CICS/IMS

J2C

CICS/IMS Java Data Binding

Show All Wizards.

Figure 4-1 Rational Application Developer J2C wizard

The Java methods generated by the wizard include support for code page conversion and conversion between Big and Little Endian data types (Figure 4-2).

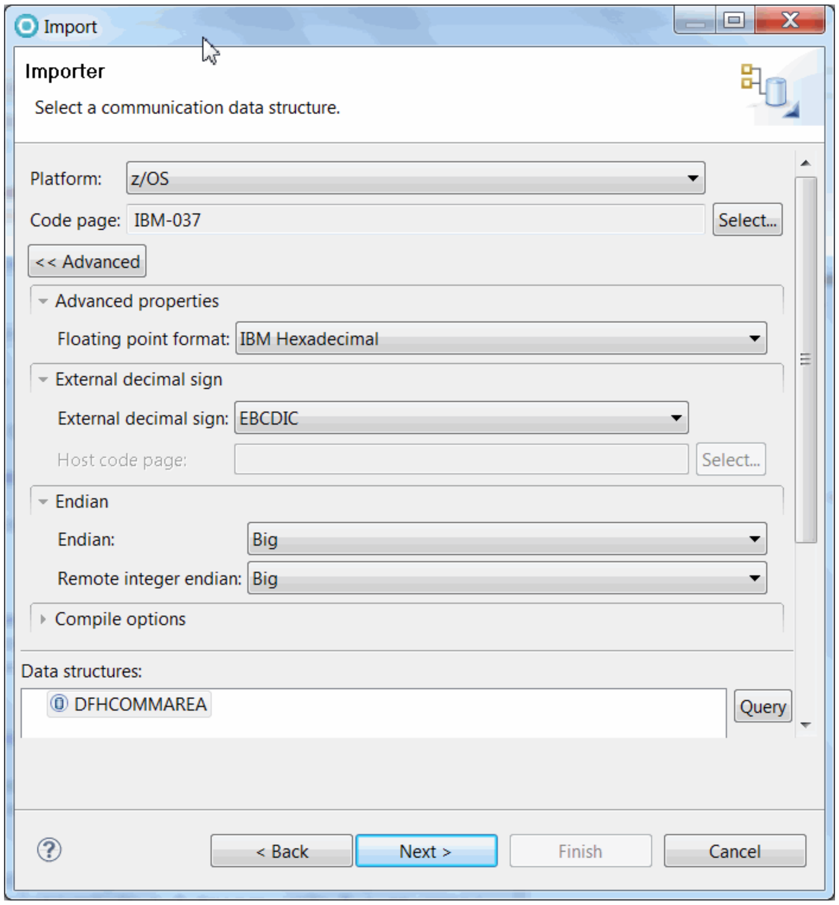


Figure 4-2 Rational Application Developer J2C wizard import options

Record Class Generator utility

Another option for generating a Java class with `get` and `set` methods for fields from an existing record layout is the JZOS Assembler and COBOL Record Generator utility that is part of the IBM experimental JZOS batch toolkit for z/OS SDKs (see Example 4-4). This utility runs on z/OS and can be used to generate Java classes from COBOL copybooks and Assembler DSECTs.

Example 4-4 Snippet of JCL executing the RecordClassGenerator utility

```
//* Generate a .java file for each copybook
//JAVA EXEC PROC=EXJZOSVM,VERSION='50'
//MAINARGS DD *
com.ibm.jzos.recordgen.cobol.RecordClassGenerator
  bufoffset=false
  package=com.ibm.ats.cobol.records
  outputDir=~/cobgen
//SYSADATA DD DSN=&&ADATA,DISP=(OLD,DELETE)
```

For more information, see the “IBM Experimental Version of the JZOS Batch Toolkit and a Cookbook for z/OS SDKs” page in the IBM developerWorks Communities section:

http://ibm.co/1vRC3f7

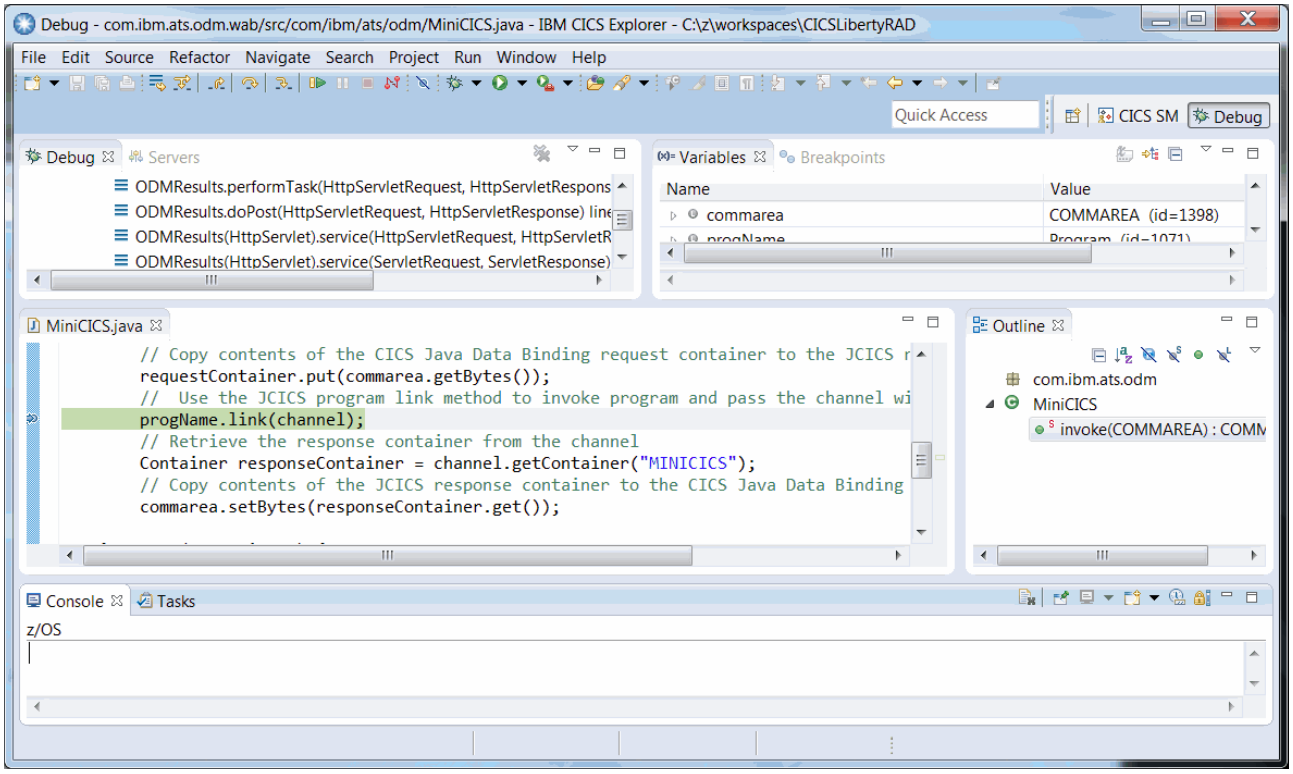
Note: Notice the Java statement in Example 4-2 on page 37 where the `responseContainer` was “moved” to the `COMMAREA` using a constructor method. This was required because this Java class was generated by the JZOS Record Generator utility, which did not generate a set method that accepts a byte array for updating the entire record. Only set methods for individual fields are generated. The Rational Application Developer J2C wizard does generate all of required methods.

You can find details about both of these techniques in Chapter 8 of the Redbooks publication titled *IBM CICS and the JVM server: Developing and Deploying Java Applications*, SG24-8038.

Note: Bean Validation 1.0 is one the features supported by the CICS Liberty JVM server, but neither of these two methods include bean validation annotations. If you want to include bean validation support, the annotations must be added manually.

4.2.2 Debugging Java in CICS Liberty JVM server

Both the CICS TS Explorer and Eclipse integrated development environments (IDEs) provide means to debug remote Java applications by stepping thought lines of code, setting breakpoints, inspecting the contents of variables and so on. This technique can be used to also debug Java applications running in CICS Liberty JVM server. Figure 4-3 shows remotely debugging a CICS Liberty JVM server application in CICS TS Explorer.



ODMResults.performTask(HttpServletRequest, HttpServletRespons
ODMResults.doPost(HttpServletRequest, HttpServletResponse) line
ODMResults(HttpServlet).service(HttpServletRequest, HttpServletR
ODMResults(HttpServlet).service(ServletRequest, ServletResponse)

Name	Value
commarea	COMMAREA (id=1398)
progName	Program (id=1071)

```
// Copy contents of the CICS Java Data Binding request container to the JCICS r
requestContainer.put(commarea.getBytes());
//  Use the JCICS program link method to invoke program and pass the channel wi
progName.link(channel);
// Retrieve the response container from the channel
Container responseContainer = channel.getContainer("MINICICS");
// Copy contents of the JCICS response container to the CICS Java Data Binding
commarea.setBytes(responseContainer.get());
```

com.ibm.ats.odm
MiniCICS
invoke(COMMAREA) : COMM

z/OS

Figure 4-3   Debug perspective of a Java applications running in CICS Liberty JVM server

The setup for a remote debugging session for CICS Java from a developer's IDE is the same as when debugging on any other remote platform. Remote debugging can be very useful when migrating an existing application to CICS Liberty JVM server or the development of a new application for CICS Liberty JVM server.

Note: To enable remote debugging of a CICS Liberty server the following statement must be included in the JVM profile of the CICS Liberty JVM server resource definition:

```
-agentlib:jdwp=transport=dt_socket,server=y,address=<port>
```

4.3 Developing new applications using other Liberty features

There are Liberty features that are unique to CICS TS. Others provide functions that are used by CICS Liberty JVM server for better integration with CICS Transaction Server for z/OS and IBM CICS Transaction Server for z/OS Value Unit Edition.

4.3.1 CICS Liberty JVM server Java Database Connectivityoptions

CICS Liberty JVM server supports two different features for accessing Java Database Connectivity (JDBC) data sources.

- The first JDBC feature (cicsts:jdbc-1.0) is unique to CICS Liberty JVM server. It is tightly integrated with CICS TS for using the existing CICS TS DB2 connection resource (the DB2CONN resource). This feature supports only a DB2 local connection (JDBC Type 2).
- The second JDBC feature (jdbc-4.0) is supplied by IBM WebSphere Application Server for z/OS Liberty profile and supports remote connections (JDBC Type 4).

Note: Use of the JDBC feature provided in the base Liberty product is discouraged for local or JDBC Type 2 connections in CICS Liberty JVM server.

Local Connections (JDBC Type 2)

The CICS TS JDBC connection feature (cicsts:jdbc-1.0) is used when an application accesses a local DB2. The access is automatically included in a logical unit work managed by the CICS TS transaction manager. Commits and rollbacks of updates made to DB2 are automatically coordinated with commits and rollbacks of other CICS TS resources with no additional coding required. Security for these applications is also managed by the CICS TS DB2 connection definition.

Remote Connections (JDBC Type 4)

CICS TS Liberty uses the Liberty JDBC feature (jdbc-4.0) for accessing remote DB2 subsystems (JDBC Type 4). Java applications that use this feature must use the Java Transaction API (JTA) to integrate commits and rollbacks for remote JDBC system with CICS TS transaction management. This allows a remote DB2 connection to participate in the same global transaction as other CICS TS resource, such as access to VSAM files, transient data queues, temporary storage queues, and so on. It also supports other JDBC drivers, such as Derby.

Note: The **`-DISPLAY DDF`** command display provides the information required to configure JDBC Type 2 and Type 4 connections. The `LOCATION` value provides the location name, and `DOMAIN` and `TCPPORT` values provide the server name and port number:

```
DSNL080I  #DI2C DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I DSNV10P2           GBIBMIYA.IYCWZDBO -NONE
DSNL084I TCPPORT=40100 SECPORT=30100 RESPORT=50101 IPNAME=-NONE
DSNL085I IPADDR=::9.20.5.0
DSNL086I SQL    DOMAIN=winmvs2c.hursley.ibm.com
DSNL086I RESYNC DOMAIN=winmvs2c.hursley.ibm.com
DSNL089I MEMBER IPADDR=::9.20.5.0
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

4.3.2 JDBC connection options

There are two ways to identify the DB2 system to which the application is to connect:

- Use a JDBC-provided DriverManager by providing a Uniform Resource Locator (URL) to provide connection properties.
- Use a data source by doing a Java Naming and Directory Interface (JNDI) lookup of the data source name to obtain connection properties at execution time.

DriverManager JDBC connections

The driver manager technique uses one of two types of URLs to identify the target DB2 subsystem and optional connection properties:

- The default URL (see Example 4-5) does not provide an explicit DB2 location name and no properties and defaults to the DB2 system are specified by the CICS TS DB2CONN resource.

Example 4-5 Example of connecting to the default DB2 subsystem using a Default UR

```
Connection connection = DriverManager.getConnection("jdbc:default:connection");
```

- The other URL format provides a DB2 location name in the URL (see Example 4-6). The location name can be the location name of the local DB2 subsystem or the location name of a remote DB2 subsystem if DB2 distributed data facility (DDF) has been configured between the local and remote DB2 subsystem.

Example 4-6 Example of requesting an explicit connection to DB2 subsystem by location name

```
Connection connection = DriverManager.getConnection("jdbc:db2os390:location");
```

Note: In a Liberty JVM server, DriverManager connections require the enablement of the CICS TS JDBC (cicsts:jdbc-1.0) Liberty feature.

DataSource JDBC connections

The DataSource method allows an application to use the Java Naming and Directory Interface (JNDI) to look up data sources defined in the Liberty `server.xml` file. A special JNDI name of `jdbc/defaultCICSDataSource` (see Example 4-7) is used by the CICS TS JDBC feature for CICS TS JDBC connections.

Example 4-7 JNDI lookup for the default CICS TS JDBC data source

```
context = new InitialContext();
dataSource = (DataSource) context.lookup(jdbc/defaultCICSDataSource);
Connection connection = dataSource.getConnection();
```

Connections to remote DB2 subsystems use the JNDI name of the data source defined in the `server.xml` file (see Example 4-8).

Example 4-8 JNDI lookup for a remote JDBC data source

```
context = new InitialContext();
dataSource = (DataSource) context.lookup(jdbc/myDataSource);
Connection connection = dataSource.getConnection();
```

Example 4-9 is a snippet of the `server.xml` file that show the JDBC JNDI configuration required for these examples.

Example 4-9 Contents of server.xml file related to JDBC

```
<featureManager>
            .....
       <feature>cicsts:jdbc-1.0</feature>
       <feature>jdbc-4.0</feature>
             .....
   </featureManager>
.......
<dataSource jndiName="jdbc/MyDataSource">
     <jdbcDriver libraryRef="defaultCICSDb2Library"/>
     <properties.db2.jcc databaseName="DSNV10P2" driverType="4"
        password="{xor}Lz4sLCgwLTs=" portNumber="40100"
        serverName="winmvs2c.hursley.ibm.com" user="DB2USER”/>
</dataSource>

<cicsts_dataSource id="defaultCICSDataSource"
jndiName="jdbc/defaultCICSDataSource"/>

<cicsts_jdbcDriver id="defaultCICSJdbcDriver" libraryRef="defaultCICSDb2Library"/>

<library id="defaultCICSDb2Library">
         fileset dir="/usr/lpp/db2/jdbc/classes" includes="db2jcc4.jar
           DB2jcc_license.jar”/>
        <fileset dir="/usr/lpp/db2/jdbc/lib" includes="libdb2jcct2zos4_64.so"/>
</library>
```

Note: The `cics_ts_jdbcDriver` and library elements are required for both DriverManager and DataSource connections. They identify the libraries where the JDBC driver components reside and the Java JAR files and shared object library to be used.

Part 3

Mobile devices

This part introduces the ability to expose existing CICS programs to mobile devices as part of a new workload. It describes two key mechanisms by which to do this, both of which involve hosting Java-based transformation services in CICS.

These two mechanisms involve the use of CICS Liberty JVM server, Java in CICS, and the JavaScript Object Notation (JSON) and XML data serialization formats. CICS Liberty JVM server allows standard Java components to be deployed to CICS, and CICS Java allows CICS web services to run in a Java-based pipeline.

This section includes the following chapters:

5 Connecting mobile devices to CICS Transaction Server

This chapter is about general considerations when using mobile devices to interact with IBM CICS TS applications. The two chapters that follow covers specific CICS TS technologies for connecting mobile devices when using IBM CICS Transaction Server for z/OS Value Unit Edition.

This chapter includes the following topics:

5.1 Mobile devices and IBM CICS Transaction Server for z/OS Value Unit Edition

For many years, CICS TS has been capable of hosting programs that can be called from outside CICS TS. Technologies such as the CICS TS Transaction Gateway, CICS TS Web Support, and CICS TS web services have enabled integration of CICS TS assets in a heterogeneous computing environment. The clients of these services could be dumb terminals, web browsers, peer servers, or even mobile devices.

Exposing existing CICS TS assets to new types of clients can be an example of a *new workload*, particularly if the client is a mobile device, or other system of engagement. There are many technologies that can be used to connect a mobile device to CICS TS, some of which involve hosting transformation services in CICS TS. The transformation service qualifies as a candidate for approval to be hosted in IBM CICS Transaction Server for z/OS Value Unit Edition if it meets these conditions:

- Enables a new type of client (that is, a new workload) to call a program in CICS TS
- Is hosted in a JVM server in CICS TS
- Links to a target application program that is hosted in a regular CICS TS region

These are examples of transformation services that enable connectivity:

- Transforming data in JSON format to and from CICS TS application data format
- Transforming data in XML format to and from CICS TS application data format

Figure 5-1 illustrates a mobile device being used to access CICS TS. A transformation service is hosted in the Value Unit Edition and provides access to applications from another CICS TS region.

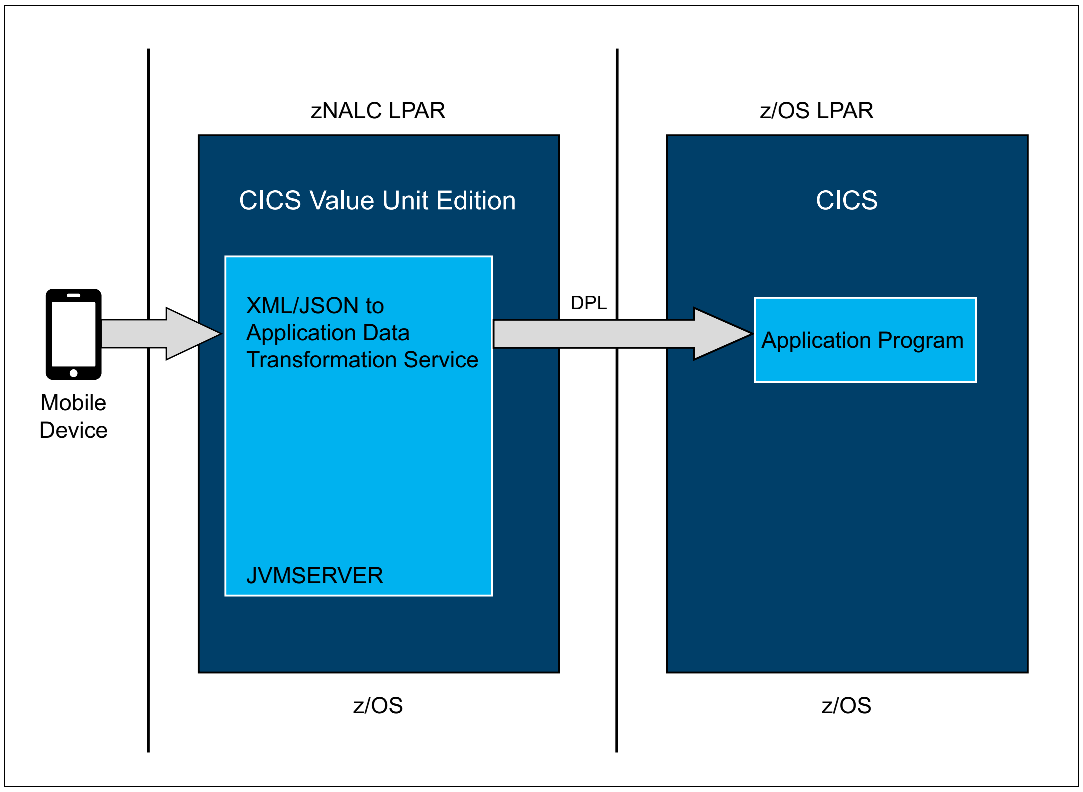


Figure 5-1 Hosting a transformation service that links to an existing CICS TS program

Two major connectivity options exist that can satisfy the criteria:

- Hosting transformation services in an IBM WebSphere Application Server Liberty profile environment in CICS TS. This can involve using standard Java data manipulation technologies, such as JAX-RS and JAX-WS. The WebSphere Application Server Liberty profile is hosted within a JVM server in CICS TS. Therefore, it is candidate for approval to be deployed to the Value Unit Edition.
- Hosting CICS TS integrated transformation services in a JVM server. This involves hosting a `WSBind` file in a Java-based pipeline in CICS TS. Because the pipeline meets this hosting criterion, it a candidate for approval to be deployed to the Value United Edition.

These two techniques are described further in the two chapters that follow:

- Chapter 6, “Mobile devices and CICS Liberty JVM server” on page 59
- Chapter 7, “Mobile devices and CICS TS Java” on page 67

The remainder of this chapter reviews issues that are common to both scenarios.

Note: Other techniques exist to connect a mobile device to CICS TS, but the associated workload is less likely to be approved to be hosted in IBM CICS Transaction Server for z/OS Value Unit Edition. The techniques presented in this book are candidates for approval for use with the Value Unit Edition.

If you’re interested in alternative connectivity, you could consider using any of these options:

- IBM WebSphere Liberty z/OS Connect
- CICS Web Support and the 3270 Web Bridge
- CICS native web services implementation
- CICS Transaction Gateway
- Connectivity through IBM MQ
- Any other mechanism that can send work to CICS

Mobile devices, and the intermediate servers that they communicate with, are general purpose computing devices. With sufficient effort, they can be made to communicate by using any protocol that is supported by CICS.

5.2 Use of mobile devices with CICS TS

The revolution in mobile computing is a significant opportunity for CICS TS based organizations. By extending existing enterprise applications to a mobile platform, a business can capitalize on its existing investment without the need to develop an entirely new solution to support mobile services. In addition, a line of business can provide service to users who increasingly expect to be able to interact with an organization byusing their mobile phones.

As a platform, these are the primary benefits offered by CICS TS with mobile devices:

- Provides reuse of existing enterprise services

  Existing assets and investments can be reused as part of a mobile application. Any existing investment in web services can be leveraged for mobile clients.

- Provides simplified access to enterprise data

  CICS TS supports access to data by using two popular data serialization formats: XML (Extensible Markup Language) and JSON (JavaScript Object Notation). JSON is a particularly popular data format for use from mobile devices.

- CICS TS already operates at the heart of the enterprise

  Hosting mobile applications within CICS TS brings them closer to the enterprise data that they are accessing. This minimizes application path lengths and improves response time.

- Adopts a RESTful architectural style for service delivery

  A RESTful architectural style is one where the target resource and the operation to be performed against it are defined by a combination of a well-structured Uniform Resource Identifier (URI) and one of the four Hypertext Transfer Protocol (HTTP) methods (`GET`, `POST`, `PUT`, and `DELETE`). This architectural approach is favored among developers of applications for mobile devices.

- Provides capacity to manage mobile workloads

  Customers around the world use CICS TS TS to host hundreds of millions, and in some cases billions, of transactions per day. CICS TS workload management provides a robust and scalable platform that is suitable for supporting the heaviest of mobile workloads.

5.3 Accessing services by using XML and JSON

There are two common standards-based technologies for cross-platform data serialization: Extensible Markup Language (XML) and JavaScript Object Notation (JSON). CICS TS supports them both. It is likely that workload initiated from a mobile device will use one of these formats.

The two formats share much in common. They both provide a way to describe structured data in an interoperable text-based form. But their differences make them more or less suitable for particular use scenarios. Either format can be used to serialize data when contacting CICS TS from a mobile device, either directly or indirectly, via an intermediate gateway, mediation service, or server.

5.3.1 Extensible Markup Language (XML)

XML is readily recognizable by the presence of angle brackets in the serialized data. XML tags wrap each atomic data unit. The size of these tags can be a significant proportion of the total size of the serialized data, which is the main cause of XML's reputation as a verbose (therefore inefficient) data format. In Example 5-1 on page 51, 160 characters of markup data are used, compared to 85 characters of application data.

XML-based web services have been supported in CICS TS Transaction Server since version 3. They involve a Web Service Description Language (WSDL) document that describes the data format for a service in an interoperable form. Client programs can be written or generated to call the XML-based web service by using the information from the WSDL document.

The XML representation of the data is encapsulated in a SOAP message. SOAP is the messaging protocol. A SOAP message can optionally contain an extensible list of headers in addition to the XML application data. A family of related specifications has evolved for SOAP and WSDL to allow for sophisticated data exchange models, including such qualities of service as transactionality, identity propagation, and dynamically reconfigurable addressing.

SOAP implements its own error format, the SOAPFault. Applications pass error information to each other by using SOAPFault messages.

XML-based web services are widely adopted in many industries and in many programming environments. They remain a favorite technology for interoperability scenarios.

Example 5-1 XML data example with 160 characters of markup, 85 characters of application data

```
<contact xmlns="http://example.org/contacts">
  <name>International Business Machines Corp.</name>
  <address>
    <street>1 New Orchard Road</street>
    <city>Armonk</city>
    <state>NY</state>
    <code>10504-1722</code>
  </address>
  <phone>800-426-4968</phone>
</contact>
```

5.3.2 JavaScript Object Notation (JSON)

JSON is a relatively young technology compared to XML. It offers an alternative method for serializing data, characterized by the presence of curly braces in the data. It generally serializes data to a shorter form than the equivalent XML, which has led to its reputation as an efficient data format. In 5.3.3, “Key differences between XML and JSON” on page 52, 81characters are used for markup of the same application data that required 160 characters for XML markup in Example 5-1.

Example 5-2 Example JSON data, 81 characters of markup, 85 characters of application data

```
{
  "name": "International Business Machines Corp.",
  "address": {
     "street": "1 New Orchard Road",
     "city": "Armonk",
     "state": "NY",
     "code": "10504-1722"
}
"phone": "800-426-4968"
}
```

JSON is particularly associated with the JavaScript programming language, which is popular as a language for mobile devices. It is often the data format of preference among developers of mobile applications. CICS TS has supported JSON-based web services since version 4 release 2, through use of the CICS TS Feature Pack for Mobile Extensions.

JSON does not have an equivalent to the WSDL document used in XML, so it can be difficult to document the characteristics of a JSON interface in a formal fashion. A JSON schema language exists for defining the syntax of JSON data, but it is not widely implemented throughout the industry.

JSON does not have the wealth of supporting specifications that are available for XML web services. This can limit its usefulness for some tasks but makes it more efficient for others.

JSON enables (but does not require) a Representational State Transfer (RESTful) interface to data. Typical CICS TS programs implement a remote procedure call that accepts input, performs an action, and returns output. This pattern is referred to as *request-response*.

A RESTful interface is quite different. It is data-driven, where each data fragment is referenced through its own URI, and the only actions that can be performed on the data are **`Create`**, **`Read`**, **`Update`**, and **`Delete`**. These actions map to the underlying HTTP methods of `POST`, `GET`, `PUT`, and `DELETE`. This concept is rather exotic under CICS TS, but it is widely used among JSON service providers. The association between JSON and RESTful services is sufficiently strong that the terms are sometimes used interchangeably.

5.3.3 Key differences between XML and JSON

The following list explains the key differences between the two formats, as implemented in CICS TS:

- The content of a SOAP message is XML data, whereas a JSON message contains JSON data. JSON and XML are different encoding mechanisms for describing structured data.
- JSON tends to be a more efficient encoding mechanism, so a typical JSON message is smaller than the equivalent XML message.
- JSON is easy to integrate in JavaScript applications, but XML is not. This difference makes JSON a preferred data format for many mobile application developers.
- SOAP provides a mechanism to add headers to a message and a family of specifications for qualities of service. For example, WS-Security defines sophisticated rules for securing SOAP messages, and WS-AtomicTransactions defines a distributed two-phase commit protocol. JSON does not have this mechanism. Instead, it relies on the services of the underlying HTTP network protocol. This reliance results in fewer options for securing and configuring a workload.
- SOAP web services are described by using WSDL documents. JSON web services are structured less formally. They tend to be loosely coupled and rely on informal documentation, often including examples of serialized data.
- SOAP has a larger ecosystem of related tools that can help with application development.
- SOAP web services have an explicit error format that involves using SOAPFault messages. There is no equivalent for JSON.
- SOAP web services support use of both HTTP and IBM MQ based messaging, whereas JSON requires HTTP.
- JSON web services support both RESTful and request-response driven interfaces, but SOAP supports only the request-response interface.
- The comparative performance characteristics of the two technologies are difficult to predict. The smaller payload of a JSON message does not necessarily result in a less costly or more efficient web service. The business cost is further complicated by the deployment technologies, which involves the use of IBM CICS Transaction Server for z/OS Value Unit Edition, IBM Mobile Workload Pricing for z/OS, and IBM System z Application Assist Processors.

Business requirements are likely to dictate which data serialization protocol is used in a project. In general, SOAP tends to be more suitable for server-to-server communication (for quality of service), and JSON is more suitable for mobile device-to-server communication (for ease of client-side development). In some deployment scenarios, JSON is be used between the mobile device and an intermediate server, and SOAP is used for further server-to-server communication.

5.4 CICS TS web service development strategies

Enabling a CICS TS program to be called from a mobile device often involves exposing it as a remotely callable web service.

There are two main service enablement techniques used to do this. The first is called *bottom-up*, and it involves exposing an existing asset to new callers. The second is called *top-down*, which involves implementing a new service that conforms to the requirements of an external specification. A third, hybrid scenario also exists, called *meet-in-the-middle*.

In each case, a transformation service is used to convert the application data to and from the format used for data interchange. The data interchange format can be either JSON or XML. The application data is structured in the format used by the application (for example, as described by a COBOL copybook).

5.4.1 Bottom-up service enablement

In this scenario, a web service (based on either JSON or XML) is generated from an existing program. The programmatic interface to the existing program (such as a COBOL copybook) is processed by using tools, and transformational metadata or programs are produced. The specifics of how this works and the artifacts produced vary by scenario. But the general concept is common to both: You have an existing asset that needs exposing as a web service, and the tools make this possible without changing the existing application.

In practice, this is not always achievable. There can be characteristics of an existing application that make it unsuitable for use as a web service. For example, it might have a 3270 interface, rely on the existence of a CICS TS terminal, or use shared memory for cross-program communication. A program's interface might rely on data types that are not supported by the tools, such as pointers, or redefined data structures.

In most cases, an existing program can be exposed as a web service, either without changes or with minimal additional effort.

5.4.2 Top-down service enablement

In this scenario, a web service (either JSON- or XML-based) is generated from a specification document, typically either a WSDL document or a JSON-Schema. Tools are used to generate new application data structures from that specification. A new application is then written that uses the generated data structures and interacts with the existing programs.

This generally results in a better web service than in a bottom-up approach. The interface is defined according to the requirements of the service, not the existing implementation. Services can be designed to support version control and to evolve with minimal disruption for existing callers.

An important variation of a top-down scenario involves creation of a *requester* or client program in CICS TS. This program can invoke a remote web service that is hosted outside of CICS TS. Requester mode scenarios can also be considered as part of a new workload if the transformation capability is hosted in a JVM server. One method that can be used is to have a CICS program link to a stub program that is hosted in IBM CICS Transaction Server for z/OS Value Unit Edition, and the stub can invoke the remote web service. The stub must either be hosted in a JVM server or drive a transformation service that is hosted in a JVM server to be a candidate for approval for the Value Unit Edition.

5.4.3 Meet-in-the-middle service enablement

This is a hybrid scenario that can combine the best features of both bottom-up and top-down service enablement. It involves mapping an existing program to a defined interface.

A typical meet-in-the-middle scenario involves generating a web service interface to an existing application by using bottom-up enablement techniques. This is followed by a manual modification of the generated service description documents. The modifications might include renaming fields so that they will make sense to an external developer, removing unnecessary content, adding additional validation rules, adding versioning information, and so on.

After a perfected interface is established, the existing program must be integrated with the new interface. This might involve the use of tools to map between the two formats or the generation of new application structures by using top-down techniques. The application might need to be changed to implement the new interface. Subsequent maintenance of the web service is performed top-down from the service description document.

This technique tends to involve significantly more effort than a pure bottom-up enablement strategy, so it is not popular for that reason. But it results in a better quality of web services.

5.5 IBM MobileFirst Platform Foundation and CICS TS

The recommended strategies for integrating mobile devices into a secure and high-performance CICS TS environment are described in the IBM Redbooks publication titled *Implementing IBM CICS JSON Web Services for Mobile Applications*, SG24-8161:

http://www.redbooks.ibm.com/abstracts/sg248161.html

A key consideration includes the use of IBM MobileFirst Platform Foundation (formerly known as IBM Worklight®), which provides a comprehensive platform on which to build, test, run, and manage mobile web applications. It can help reduce both application development and maintenance costs, improve time to market, and enhance mobile application governance and security.

Figure 5-2 on page 55 illustrates a MobileFirst server being used to mediate between the mobile devices and CICS TS.

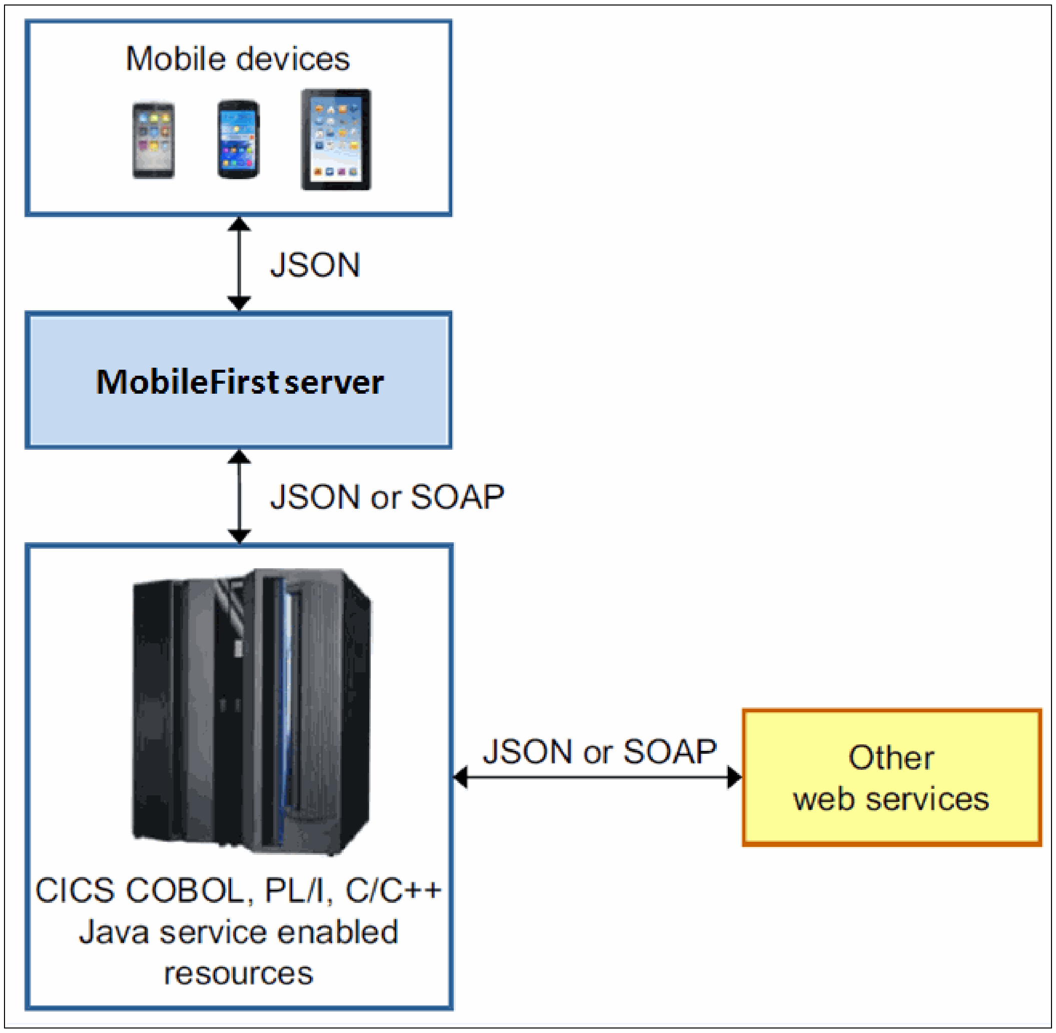


Figure 5-2 IBM MobileFirst server as an intermediate node between mobile devices and CICS TS

The MobileFirst server acts as an aggregation point, allowing many devices to share a single connection to CICS TS. It also provides device abstraction services for the application developer, helps ensure the physical security of the devices, and can provide protocol conversion services for CICS TS.

The mobile devices can communicate with the MobileFirst server by using JSON format data, and the MobileFirst server can communicate with CICS TS by using whatever mechanism is most convenient, including both JSON- and SOAP-based web services.

The MobileFirst capabilities for mobile application security are explained in the IBM Redbooks publication titled *Securing Your Mobile Business with IBM Worklight*, SG24-8179:

`http://www.redbooks.ibm.com/abstracts/sg248179.html`

5.6 IBM DataPower and CICS TS

An IBM WebSphere DataPower® appliance is often deployed alongside CICS TS to secure web services workloads.

DataPower can secure mobile traffic before it arrives at CICS TS and validate the JSON and XML data at high speeds, acting as a form of firewall. It can also be used in combination with IBM MobileFirst to provide authentication services for mobile devices.

The combination of DataPower and MobileFirst to secure mobile workloads is described in detail the IBM Redbooks publication titled *Securing Your Mobile Business with IBM Worklight*, SG24-8179, cited previously.

Figure 5-3 illustrates DataPower being used to authenticate users and secure communication between mobile devices and the MobileFirst server.

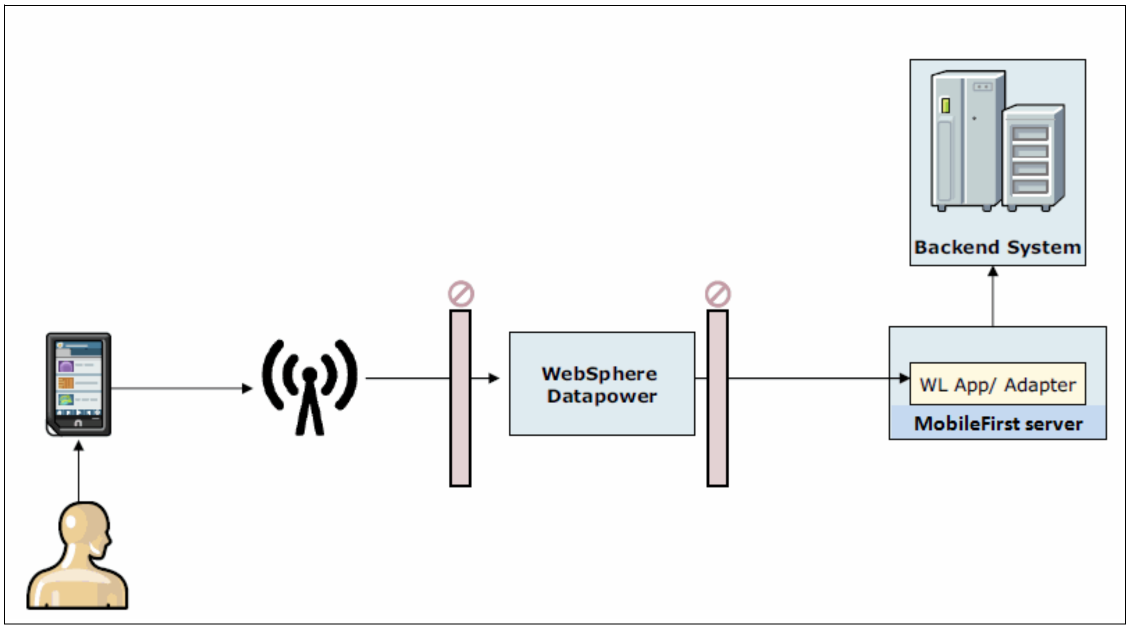


Figure 5-3 IBM DataPower appliances authenticate user sand protect communication to the MobileFirst server

The DataPower capabilities for integrating and securing web services for CICS TS are explained in the IBM Redbooks publication titled *Set Up Security and Integration with the DataPower XI50z for zEnterprise*, SG24-7988:

http://www.redbooks.ibm.com/abstracts/sg247988.html

5.7 Configuration for high availability

Configuring a CICS infrastructure for high availability involves two primary techniques:

- Using TCP/IP load balancing to distribute work across several routing regions
- Using a distributed program link to route work to a suitable application-owning region

Note: Additional techniques may be available for deployment scenarios that involve the use of WebSphere Application Server Liberty profile.

Figure 5-4 illustrates a workload being balanced across a series of CICS routing regions, each of which hosts a transformation service for new workload and routes the work to a group of application-owning regions (AORs) in another LPAR.

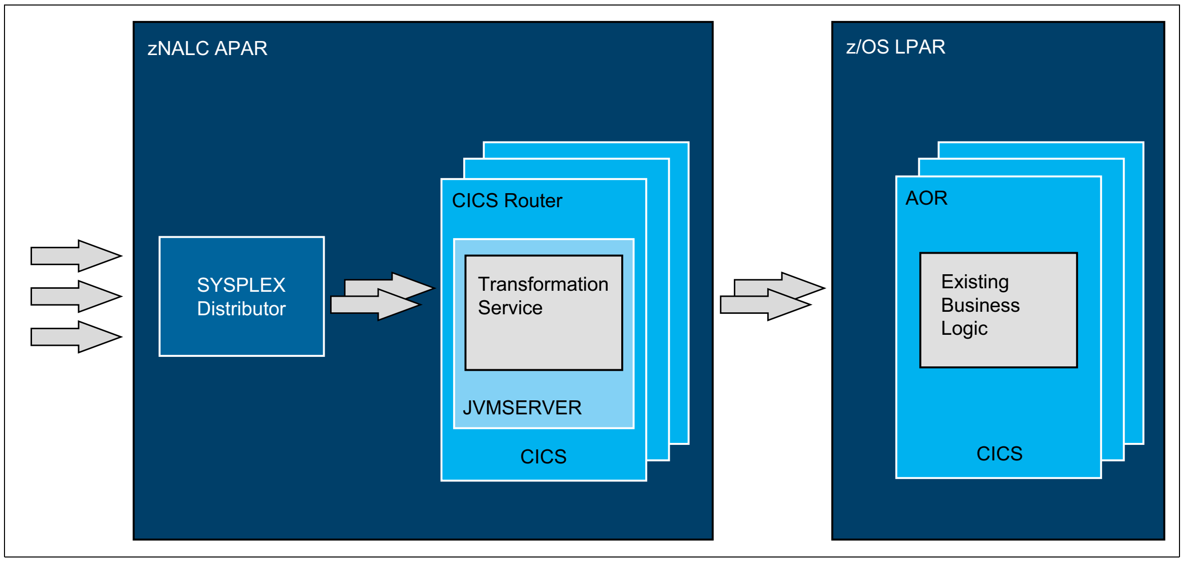


Figure 5-4 Workload balancing across multiple regions

These techniques are suitable for use in both of the implementation scenarios described in Chapter 6, “Mobile devices and CICS Liberty JVM server” on page 59 and in Chapter 7, “Mobile devices and CICS TS Java” on page 67.

The IBM Redbooks publication titled *CICS Web Services Workload Management and Availability*, SG24-7144, explores the CICS architectures for supporting high availability of web services:

http://www.redbooks.ibm.com/abstracts/SG247144.html

The two chapters that follow investigate two different technologies that are suitable for hosting Java-based transformation services:

- Chapter 6, “Mobile devices and CICS Liberty JVM server” on page 59
- Chapter 7, “Mobile devices and CICS TS Java” on page 67

6

Mobile devices and CICS Liberty JVM server

This chapter describes hosting Java-based web services in IBM CICS Transaction Server (TS), using CICS Liberty JVM server. These web services can support either SOAP or JSON data encoding formats, and they use JAX-WS and JAX-RS technologies.

This information builds on the information in Chapter 5, “Connecting mobile devices to CICS Transaction Server” on page 47, and we cover the following topics:

6.1 Hosting transformation services in CICS Liberty JVM server

At the time of writing, there are two main options for hosting a data transformation service in CICS Liberty JVM server within CICS TS. The first uses a technology called the Java API for XML Web Services (JAX-WS), and the second uses the Java API for RESTful Web Services (JAX-RS). These technologies implement SOAP and JSON for Java applications.

Figure 6-1illustrates a transformation service wrapper hosted in CICS Liberty JVM server. It is used to expose an existing application to a new workload initiated from mobile devices.

In both scenarios, a generated Java interface is hosted in a CICS Liberty JVM server environment, and custom control logic is written to interface a Java program with the existing CICS TS programs.

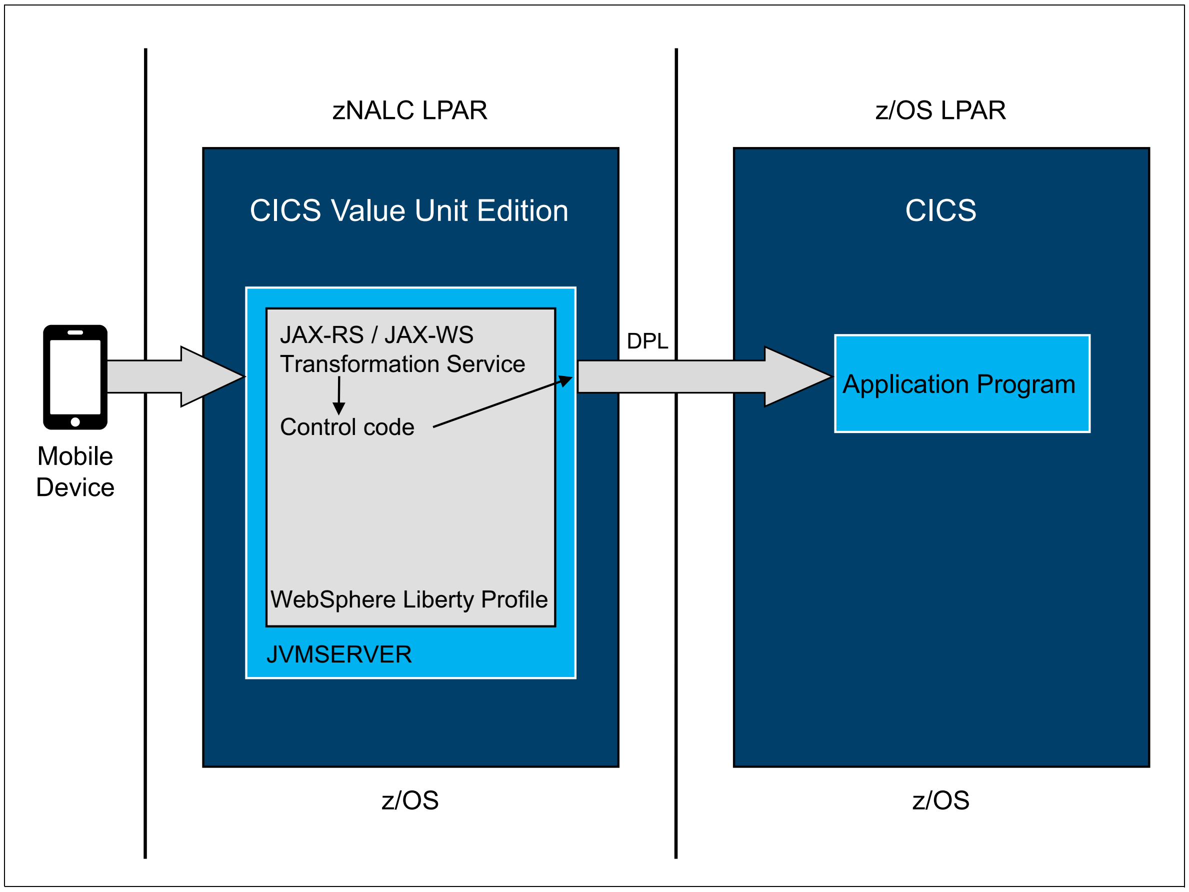


Figure 6-1 Hosting JAX-WS or JAX-RS services in a CICS Liberty JVM server environment

6.1.1 Java API for XML Web Services (JAX-WS)

The Java API for XML Web Services (JAX-WS) is a standard component of Java. It is available on all platforms supported by Java, and it allows Java applications to implement SOAP-based web services.

It supports both bottom-up service enablement patterns (where a WSDL interface is generated from an existing Java class) and top-down enablement patterns (where Java classes are generated from an existing WSDL document). (See 5.4.1, “Bottom-up service enablement” on page 53 and 5.4.2, “Top-down service enablement” on page 53 for more information.) It also supports Requester mode scenarios, where a CICS TS Java program is used to invoke a remote SOAP web service.

JAX-WS provides the tools required to bind Java programs to WSDL, and the runtime transformation service required to perform the data transformations. This transformation service is implemented in Java, so is a candidate for approval for IBM CICS Transaction Server for z/OS Value Unit Edition.

Provider mode JAX-WS applications are a fully supported component of the CICS Liberty JVM server environment in CICS TS. Requester mode applications are also supported under CICS Liberty JVM server, but are of less value due to the inability to link to a Liberty application from a traditional CICS TS program. Requester mode applications can be hosted in a regular non-Liberty JVM server in CICS TS.

Characteristics of a JAX-WS application in CICS Liberty JVM server

A JAX-WS application is, by definition, a Java application. This in turn implies that a JAX-WS application cannot use bottom-up development techniques to expose an existing non-Java CICS TS application as a SOAP web service. There will always be some form of meet-in-the-middle technique required to wrapper an existing non-Java program with a JAX-WS interface.

This will require additional application development effort compared to some other service enablement strategies, but it can be worth considering. There are several significant benefits:

- The JAX-WS service is a candidate for approval to be hosted in IBM CICS Transaction Server for z/OS Value Unit Edition.

  This can be relevant for business reasons, independent of the technical considerations.

- JAX-WS provides a complete and concise mapping of WSDL to Java.

  If you've struggled with mapping complex WSDL documents and XML schema into COBOL (or other native programming languages), JAX-WS can provide an alternative path to Service Enablement.

  Some XML constructs are inherently difficult to map into COBOL style data. Examples include inheritance hierarchies, recursive data structures, and cross-references. Java is more naturally suited to representing such information.

- JAX-WS is a cross-platform technology

  JAX-WS is a standard part of Java, and is supported by many vendors. The skills required to create JAX-WS services are not CICS TS specific.

An example JAX-WS application deployed using CICS Liberty JVM server is available in the CICSdev Community on IBM developerWorks:

"JAX-WS and JAXB support in CICS TS V5.2 Liberty JVM server," a blog entry by Mark Cocker

`http://ibm.co/1gceyS0`

Example 6-1 illustrates a simple JAX-WS application.

Example 6-1 Example of a JAX-WS Hello World application

```
package org.example;
import javax.jws.*;

@WebService
public class TextService
{
  @WebMethod public String decorateText(String text)
  {
    return "You said: '" + text + "'";
  }
}
```

6.1.2 Java API for RESTful Web Services (JAX-RS)

The Java API for RESTful Web Services is also a standard component of Java but not distributed as part of the Java Software Development Kit (SDK). A reference implementation is available on all platforms that are supported by Java. It allows Java applications to implement JSON and XML-based web services, both in RESTful and request-response use patterns. In this document we will consider JAX-RS for enabling JSON connectivity.

It supports the bottom-up service enablement patterns, where a JSON interface is generated from an existing Java class (see "Bottom-up service enablement" on page 53). It also supports Requester mode scenarios, where a CICS TS Java program is used to invoke a remote JSON web service.

JAX-RS provides the runtime service that is required to perform the data transformations. This transformation service is implemented in Java, and is a candidate for approval to be hosted in IBM CICS Transaction Server for z/OS Value Unit Edition.

Provider mode JAX-RS applications are a fully supported component of the CICS Liberty JVM server environment in CICS TS. Requester mode applications are also supported under CICS Liberty JVM server, but are of less value due to the inability to LINK to a Liberty application from a traditional CICS TS program. Requester mode applications can be hosted in a regular non-Liberty JVM server in CICS TS.

Characteristics of a JAX-RS application in CICS Liberty JVM server

JAX-RS applications are very similar to JAX-WS applications, they share many of the same characteristics. For example, a JAX-RS application must be written in Java. This means (as with JAX-WS) that additional application code must be written in Java to make a JAX-RS application communicate with any non-Java CICS TS programs. A traditional non-Java CICS TS program cannot be exposed as a JSON web service using JAX-RS in a pure bottom-up fashion.

The benefits of JAX-RS include:

1. The JAX-RS service is a candidate to be hosted in IBM CICS Transaction Server for z/OS Value Unit Edition

   This can be relevant for business reasons, independent of the technical considerations.

2. JAX-RS provides a simple mechanism to expose Java programs as JSON web services.

   Customized mapping information can be stored within the Java application using a library of Java Annotations.
3. JAX-RS is a cross-platform technology

   JAX-RS is a standard part of Java, and is supported by many vendors. The skills required to create JAX-RS services are not CICS TS specific.

An example RESTful JAX-RS application deployed by using CICS Liberty JVM server is available in the CICSdev Community on IBM developerWorks:

Writing RESTful web services using a CICS Liberty JVM Server - Part 2, a blog entry by Daniel Fitzgerald

http://ibm.co/12X3B72

Example 6-2 illustrates a simple JAX-RS application.

Example 6-2 Example of a JAX-RS Hello World application

```
package org.example;
import javax.ws.rs.*;
import javax.ws.rs.core.*;

@Path("/json/example")
public class TextService2
{
  @POST
  @Path("/post")
  @Consumes(MediaType.APPLICATION_JSON)
  @Produces(MediaType.APPLICATION_JSON)
  public String decorateText(String text)
  {
    return "You said: '" + text + "'";
  }
}
```

6.2 z/OS Connect and CICS Liberty JVM server

IBM WebSphere Liberty z/OS Connect is software that enables z/OS systems such as CICS TS and IMS to participate in a Mobile computing environment. It runs within the WebSphere Application Server Liberty Profile and provides a JSON-based interface between mobile and cloud devices and back-end systems.

z/OS Connect provides an alternative mechanism for wrapping CICS TS assets with a JSON interface. However, at time of writing it cannot be hosted in CICS Liberty JVM server. This limits its relevance to an IBM CICS Transaction Server for z/OS Value Unit Edition environment.

For more information about z/OS Connect, see the IBM WebSphere Application Server for z/OS Liberty Profile web page:

http://www.ibm.com/developerworks/downloads/ws/waszosliberty/

IBM released a statement of direction for z/OS Connect as part of the CICS Transaction Server for z/OS V5.2 announcement letter. See the web page titled “IBM CICS Transaction Server for z/OS V5.2 takes service agility, operational efficiency, and cloud enablement to a new level” (IBM United States Software Announcement 214-107, April 7, 2014):

http://ibm.co/1vnTOOq

That statement reads:

> IBM intends to deliver IBM WebSphere Liberty z/OS Connect (z/OS Connect) as a common program component of WebSphere Application Server for z/OS, IMS Enterprise Suite for z/OS, CICS Transaction Server for z/OS, and CICS TS Transaction Gateway. z/OS Connect is intended to provide a simplified, secure, and scalable gateway functionality to route web, cloud, and mobile application traffic that accesses applications provided by the aforementioned z/OS products, as well as z/OS batch and z/OS UNIX System Services applications. z/OS Connect intends to offer: (i) a fast onramp interface to z/OS applications by providing a common access mechanism based on RESTful services; (ii) tools to allow a cloud or mobile developer to define secure enterprise connectivity without the need for extensive code development or knowledge of System z.

No further information was available at the time of writing, but z/OS Connect might become relevant to IBM CICS Transaction Server for z/OS Value Unit Edition at some point after the publication of this book.

6.3 Connectivity from Java to CICS TS

In both of the scenarios described, the JAX-RS or JAX-WS application interacts with the services and programs in CICS TS. They do this by using the JCICS application programming interface (API). Application code is written in Java that converts between the Java data format and the format used by other CICS TS programs. For example, the Java application might need to construct a comm area to pass to a COBOL program or interpret the contents of a CICS TS container.

The techniques for interacting with structured records from Java are described in 4.2.1, “Java access to records and their fields” on page 37.

6.4 Security considerations

Security of web services (both JSON and SOAP) is a broad topic, encompassing many considerations.

Mobile devices are usually connected to CICS TS by using an intermediate proxy. The security configuration might be different between the mobile device and the proxy and between the proxy and CICS TS. Mobile devices can benefit from the device security characteristics of IBM MobileFirst, formerly called IBM Worklight (see 5.5, “IBM MobileFirst Platform Foundation and CICS TS” on page 54), and the authentication and firewall capabilities of IBM DataPower (see 5.6, “IBM DataPower and CICS TS” on page 56).

CICS Liberty JVM server supports several mechanisms by which a proxy can be authenticated to CICS TS. These are described in 2.5, “Security” on page 20.

At the time of writing, CICS Liberty JVM server does not support the use of the WS-Security specification for SOAP-based web services nor OAuth-based authentication for mobile devices. But most of the benefits of these protocols can be experienced through the combination of IBM MobileFirst or IBM DataPower with CICS TS.

6.5 Other considerations

There are many ways to call an existing CICS TS program from a mobile device. Hosting a transformation service in CICS Liberty JVM server offers the advantage that the associated transformation work is a candidate for approval for use with IBM CICS Transaction Server for z/OS Value Unit Edition. However, other options are available, including the use of a Java pipeline as described in Chapter 7, “Mobile devices and CICS TS Java” on page 67.

One of the limitations of the CICS Liberty JVM server approach is that the transformation work is not as tightly integrated with CICS TS as with some of the other options. For example, the CICS Liberty JVM server environment listens on its own TCP/IP socket, rather than using the CICS native sockets listener. This affects the techniques that must be used to investigate problems.

The CICS Liberty JVM server environment does not use a CICS TS PIPELINE resource and does not share the same characteristics that might be familiar from that environment. For example, the CICS TS supplied handler programs that implement the various SOAP web services specifications, such as WS-AtomicTransactions and WS-Security, are not available to use. Another difference is that the CICS TS SOAPFault API is not available for native CICS TS applications to use. However, the pattern of wrapping existing CICS TS assets with a CICS Liberty JVM server hosted transformation service can be very useful.

7

Mobile devices and CICS TS Java

This chapter describes hosting Java-based web services in an IBM CICS Transaction Server (TS) Java virtual machine (JVM) server without using CICS Liberty JVM server. These web services run in a Java-based pipeline in CICS and use the CICS supplied data transformation service to interact with CICS programs.

This chapter builds on the information in Chapter 5, “Connecting mobile devices to CICS Transaction Server” on page 47, and covers the following topics:

- 7.1, “Hosting transformation services in CICS TS Java” on page 68
- 7.2, “Characteristics of CICS data transformation” on page 69
- 7.3, “The Java-based pipeline” on page 69
- 7.4, “Security considerations” on page 70
- 7.5, “Other considerations” on page 70

7.1 Hosting transformation services in CICS TS Java

CICS has had the ability to automatically transform XML into application data as part of a web service since CICS Transaction Server V3.1. The ability to host this transformation service within a JVM server resource was introduced in CICS Transaction Server V4.2. This Java-based transformation service was extended to support JSON as part of the CICS Transaction Server Feature Pack for Mobile Extensions, V1.0. The Feature Pack is available for CICS Transaction Server V4.2 and V5.1. The capability is integrated into CICS from CICS Transaction Server V5.2.

The JVM server-based environment for CICS web services is referred to as a *Java-based pipeline*. A Java-based pipeline that exposes an existing program to mobile devices is a candidate for approval for hosting in IBM CICS Transaction Server for z/OS Value Unit Edition.

Figure 7-1 illustrates a Java-based pipeline being used to expose an existing CICS program as an XML or JSON web service.

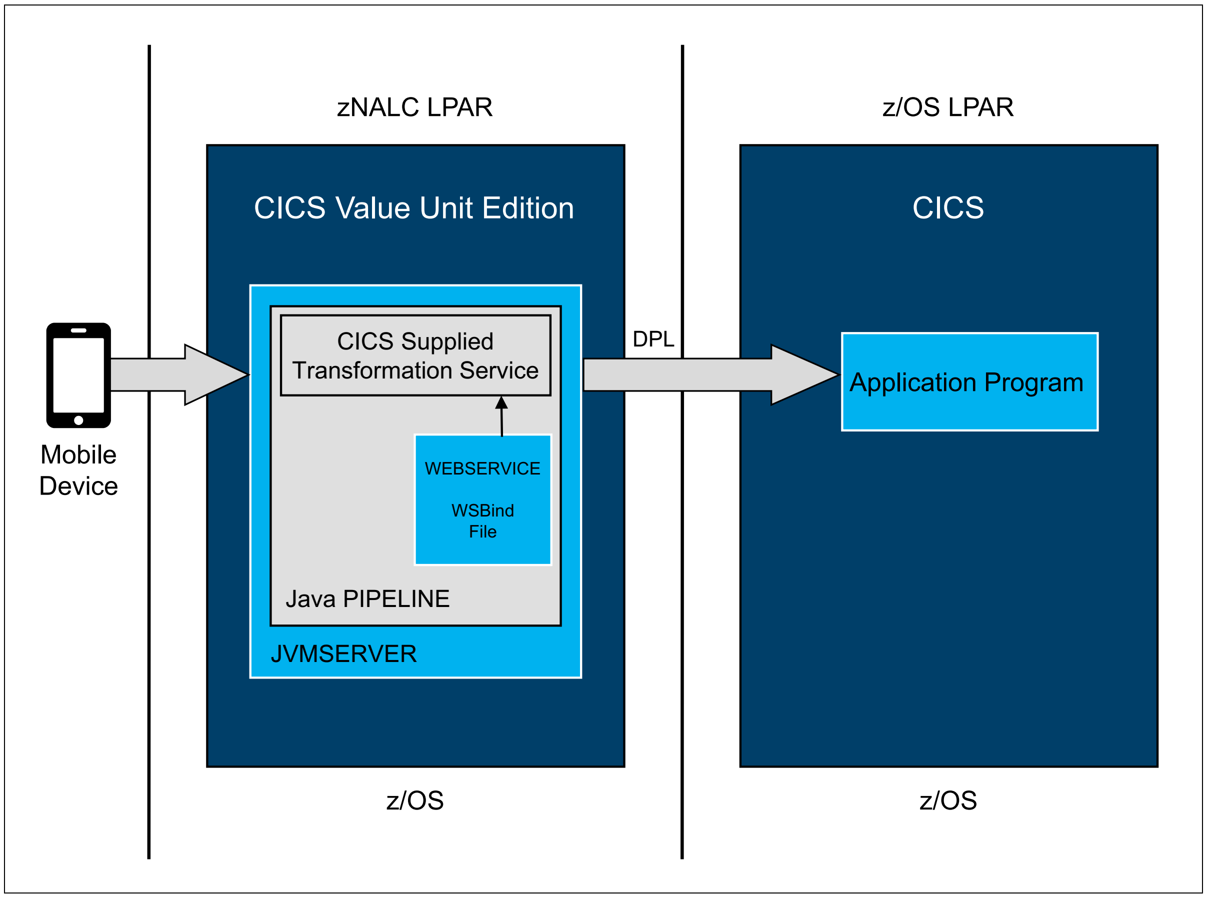


Figure 7-1 Hosting the CICS supplied transformation service in a Java pipeline

7.2 Characteristics of CICS data transformation

The Java-based CICS data transformation service facilitates the conversion of XML or JSON into structured application data and back again. It does this by using pregenerated metadata (a WSBind file) that instructs CICS in how to transform the data. The metadata is prepared in advance, using tools.

The application development aspects of how this works are described in the IBM Redbooks publication titled *Application Development for CICS Web Services*, SG24-7126:

http://www.redbooks.ibm.com/abstracts/sg247126.html

The JSON-specific characteristics of this process are described in the IBM Redbooks publication titled *Implementing IBM CICS JSON Web Services for Mobile Applications*, SG24-8161:

http://www.redbooks.ibm.com/abstracts/sg248161.html

The CICS supplied data mapping service is popular, because most existing CICS programs can be exposed as web services (either JSON or XML) without application changes and without the need for a new wrapper program to be written. The technology involved has also been adapted for other IBM products, including IBM Rational Developer for System z (which provides application development support for CICS web services), CICS Transaction Gateway (which supports hosting CICS WSBind files for JSON from version 9.1), and WebSphere Application Server Liberty profile (which supports hosting WSBind files for JSON).

The CICS data mapping technology can be used for both *bottom-up* development scenarios, and *top-down* scenarios, and it can be used in both *Provider* mode and *Requester* mode. For more informaiton about these development scenarios, see 5.4, “CICS TS web service development strategies” on page 53.

The CICS data mapping tools and the regular non-Java data transformation service have been proven in many CICS web service customer deployments since CICS Transaction Server V3.1.

7.3 The Java-based pipeline

A Java-based pipeline in CICS is very similar to a regular CICS pipeline resource (of the type that has been available for hosting web services from CICS Transaction Server V3.1). Its main differentiating characteristic is that it is implemented in Java, and is therefore eligible to be considered for approval for IBM CICS Transaction Server for z/OS Value Unit Edition.

Converting a traditional CICS pipeline to a Java-based pipeline involves:

- the configuration of a suitable JVM server in CICS
- minor changes to the pipeline configuration file
- (optionally) changing any pipeline handler programs to use Java

This can be a convenient mechanism for adapting an existing CICS web services infrastructure, though it is unlikely to perform as well as the native CICS implementation. If a JSON interface is required, consider using the capabilities of IBM z/OS Connect, see 6.2, “z/OS Connect and CICS Liberty JVM server” on page 63.

The Java-based pipeline takes XML or JSON data, and automatically converts it to structured application data, using instructions found within the WSBind file. It then LINKs to the target application program passing the transformed data as input, and finally converts the response back to XML or JSON to return to the remote caller. This can provide a significant amount of value for relatively little effort.

7.4 Security considerations

Security of web services (both JSON and SOAP) is a broad topic, encompassing many considerations. As with the CICS Liberty JVM server scenario in 6.4, “Security considerations” on page 64, both IBM MobileFirst and IBM DataPower can be used to provide additional capabilities beyond those natively available through CICS.

CICS supports a wide range of Security protocols for XML-based web services, including WS-Security, and WS-Trust, and in CICS Transaction Server Version 5.2 this is extended to include aspects of the SAML and Kerberos protocols. CICS also supports the z/OS identity propagation protocol.

General security considerations are described further in IBM Redbooks publication *SG24-7658-00 Securing CICS Web Services*. It is available here:

http://www.redbooks.ibm.com/abstracts/sg247658.html

The Identity Propagation protocol is described in IBM Redbooks publication *SG24-7850-00 z/OS Identity Propagation*. It is available here:

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

Many deployments of CICS web services include the use of an IBM DataPower appliance for authentication and as an XML firewall. A trust relationship is configured between CICS and the appliance by using a client-certified transport connection, and the user ID associated with the appliance is granted surrogate authority to assert the identity of the user. The identity of this user is passed to CICS from the appliance using WS-Security. This is probably the single most common configuration for authentication with XML-based web services for CICS: DataPower is responsible for authentication, and CICS is configured to trust DataPower to supply the identity of the user.

7.5 Other considerations

There are many ways to call an existing CICS TS program from a mobile device. Hosting a transformation service in a Java-based pipeline offers the advantage that the associated transformation work is a candidate for approval for use with IBM CICS Transaction Server for z/OS Value Unit Edition. However, other options are available, including the use of CICS Liberty JVM server as described in Chapter 6, “Mobile devices and CICS Liberty JVM server” on page 59.

One of the limitations of the Java-based pipeline is that the associated JVM server will not be suitable for hosting OSGi applications or CICS Liberty JVM server. It requires a special JVM server of its own.

The transformation service for JSON web services may be better hosted in the IBM CICS Transaction Gateway V9.1 or in WebSphere Application Server Liberty profile.

Part 4

IBM Operational Decision Manager

This section introduces the benefits of decision management for the enterprise. It explains how IBM Operational Decision Manager for z/OS allows existing IBM CICS Transaction Server COBOL and PL/I applications to use decision management as a new workload in CICS TS.

The following chapters are included in this section:

8

Decision management integrated in IBM CICS Transaction Server

This chapter introduces the concept of decision management. It also describes how IBM Operational Decision Manager for z/OS is a candidate for approval for running within IBM CICS Transaction Server for z/OS Value Unit Edition to provide decision management for CICS TS COBOL and PL/I applications.

The following topics are covered in this chapter:

8.1 Introduction to decision management

Making decisions has always been at the center of business. For instance, banks must decide who they will lend money to, insurance companies must decide who they will insure, retailers must decide when promotions will occur and who will be eligible for them. Every business must make decisions to be successful and today's businesses must make a startling number of decisions every day.

A customer's demand for a flawless, seamless, instant experience means businesses are making more decisions, they have less time to make them, and they need to get those decisions right the first time. This requires decisions that are consistent with business policy and can be made at machine speed, without manual processes and human involvement.

8.1.1 Common business decisions that require managing

There are, generally speaking, three types of decisions found in most businesses:

- Decisions that help increase revenue:
  - This type includes decisions that are used by marketing and sales to make targeted offers, based on customer profiles, demographics, and analytical models.
  - For example: Is a customer eligible for a certain promotion or a cross-sell or up-sell opportunity? Should a store discount the price of a product at the end of the day?
- Decisions about consistency and compliance with regulations:
  - This type of decision can be found in all industries, such as financial, insurance, and government sectors.
  - For example: Are there prohibitions against a customer buying a certain quantity of a product? Is a customer eligible to make a certain purchase, based on where she is located?
- Decisions that reduce and mitigate risk:
  - This type of decision includes decisions that protect the company and the customers.
  - For example: Does the customer who just filled out a loan application online meet the criteria to be approved?

Businesses must make one or all of these types of decisions many times a day and ensure that they are made correctly and consistently, according to their business policies.

8.1.2 Where most decisions are made today

The traditional approach to decision making requires a business analyst to understand the business policies and create a requirements document that defines the decisions to be made. Then, one or more software developers take the requirements and code or embed the decisions in the various application programs that support the business. The development is then followed by a lengthy testing process before the new decisions become live in production.

Unfortunately, the decisions are now hidden in the code of one or more programs, and over time as additional changes are added to the business policy, the code becomes more complex, making it difficult to change, hard to visualize and nearly impossible to manage. The decisions may change frequently or rarely and changing a program to change the decision, testing it, and getting it into production is not fast enough in today's business environment.

This scenario can be avoided by implementing a decision management solution that takes the decisions out of code and places them in a central repository. This makes the decisions more flexible, visible, auditable and manageable. This is illustrated in Figure 8-1:

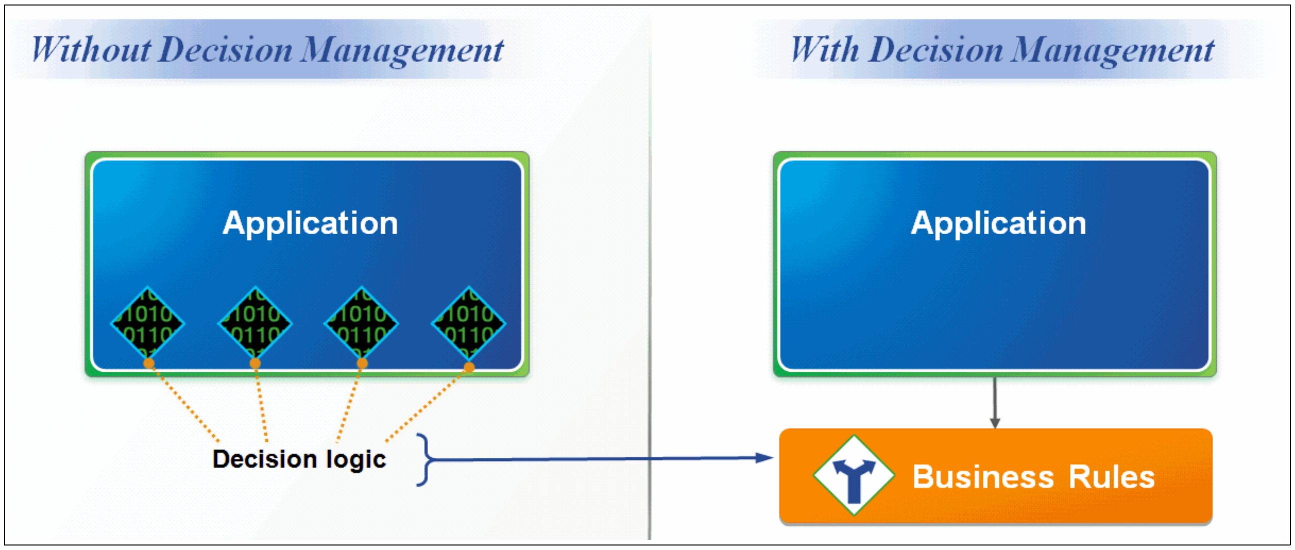


Figure 8-1 The decision logic is moved out of code and into Decision Manager

Not all decisions are equal, and some are more applicable for decision management than others. The following decisions are most appropriate for decision management:

- Decisions that must be changed frequently to support the business

  Decision management avoids costly application code changes.

- Decisions that are duplicated in multiple applications running on multiple platforms

  Decision management implements the decision once and stores it centrally to be called from multiple applications.

- Decisions that must be visible for business purposes

  Decision management allows decisions to be shared easily with lines of business or regulatory auditors.

The next section introduces IBM Operational Decision Manager for z/OS and explains how it can be used to implement decision management on IBM z/OS systems in CICS TS.

8.2 IBM Operational Decision Manager for z/OS

Operational Decision Manager for z/OS provides a smarter way of dealing with business decisions. It helps organizations gain more control over the business decisions that take place in their enterprise applications. Businesses that use Operational Decision Manager for z/OS will simplify their ability to change decisions in enterprise applications. This enables them to cut costs and cycle times, improve their agility and time to market, and enhance their documentation of business decisions, as well as the governance of those decisions.

Operational Decision Manager for z/OS delivers these advantages to enterprise users because it enables separation of the decision logic from business applications and processes.

8.2.1 Operational Decision Manager components

Operational Decision Manager for z/OS includes four main components that work together to provide a full decision management experience:

- *Rule Designer* is used as the starting point to create the model on which the business decisions are written. Rule Designer is an Eclipse-based development toolkit for business decisions. It is installed on a Microsoft Windows or Linux workstation.
- *Decision Center* is used as a team repository to govern the business decisions and to write them through a web interface. Decision Center runs on WebSphere Application Server on z/OS, Linux for System z, or a distributed environment.
- *Decision Server* is where business decisions are made. API calls are issued from COBOL and PL/I programs to the Decision Server. The Decision Server has multiple deployment options, depending on the environment the application that it is running in. These environments are described in 8.2.4, “Execute decisions by using the Decision Server” on page 80.
- *Rule Execution Server* console is a web-based graphical interface to monitor Decision Servers and to manage the deployed artifacts, such as RuleApps, rule set archives, and required libraries. It runs in either a stand-alone address space or within WebSphere Application Server for z/OS.

Figure 8-2 shows the interaction between these components. Rule Designer can create and synchronize rule projects with the Decision Center. Then, business users can use the Decision Center to modify, test, and simulate business decisions. The decisions are then published to the Decision Server by using either the Rule Designer or Decision Center.

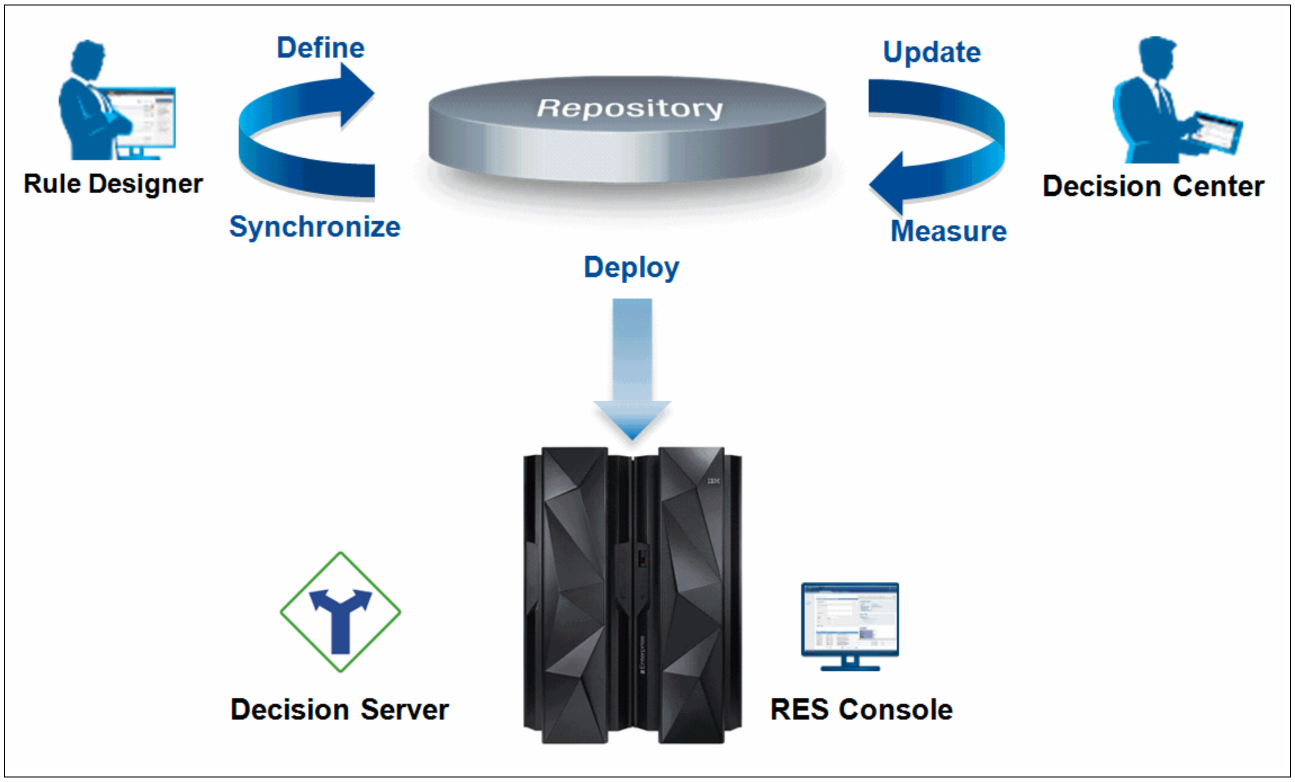


Figure 8-2 Operational Decision Manager components working together

Rule Designer must run on a Windows or Linux workstation, but Decision Center can run on multiple operating systems. This allows flexible deployment patterns, as shown in Figure 8-3.

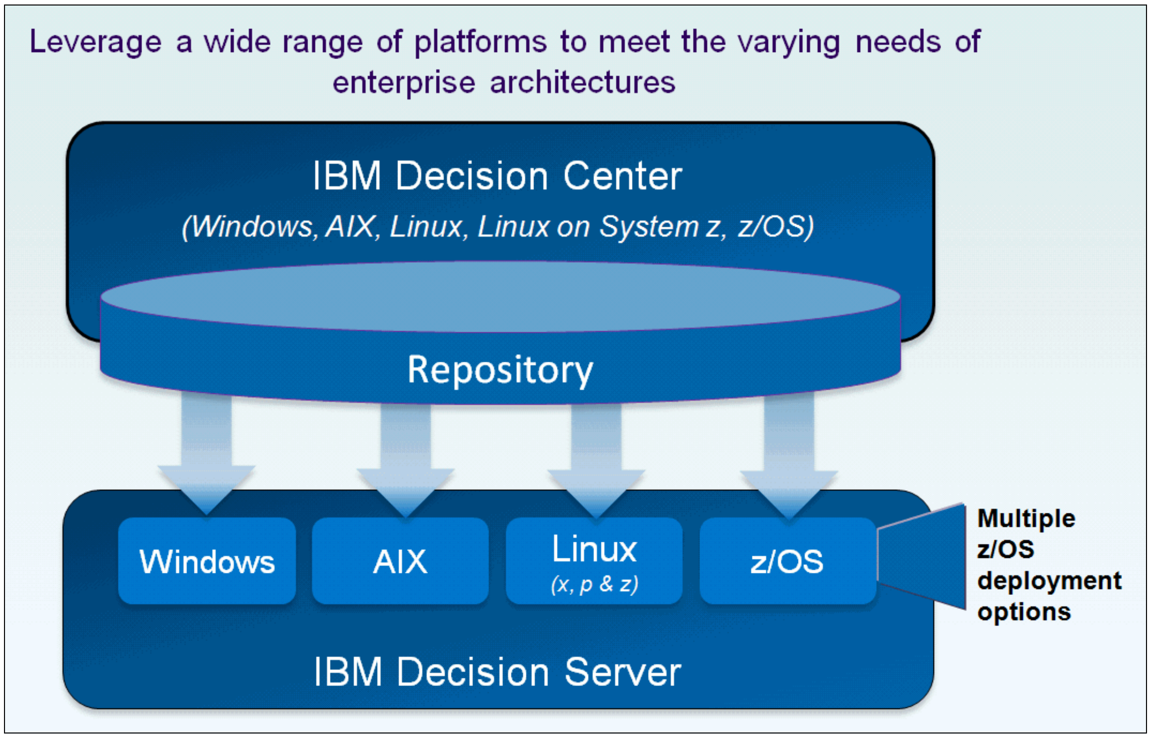

Leverage a wide range of platforms to meet the varying needs of enterprise architectures
IBM Decision Center
(Windows, AIX, Linux, Linux on System z, z/OS)
Repository
Windows
AIX
Linux
(x, p & z)
z/OS
IBM Decision Server
Multiple z/OS deployment options

Figure 8-3 Decision Center and Decision Server can run on various operating systems

> **Note:** A recommended approach when using Operational Decision Manager for z/OS is to run Decision Center on a distributed or Linux on System z platform and deploy remotely to Decision Server running on z/OS.

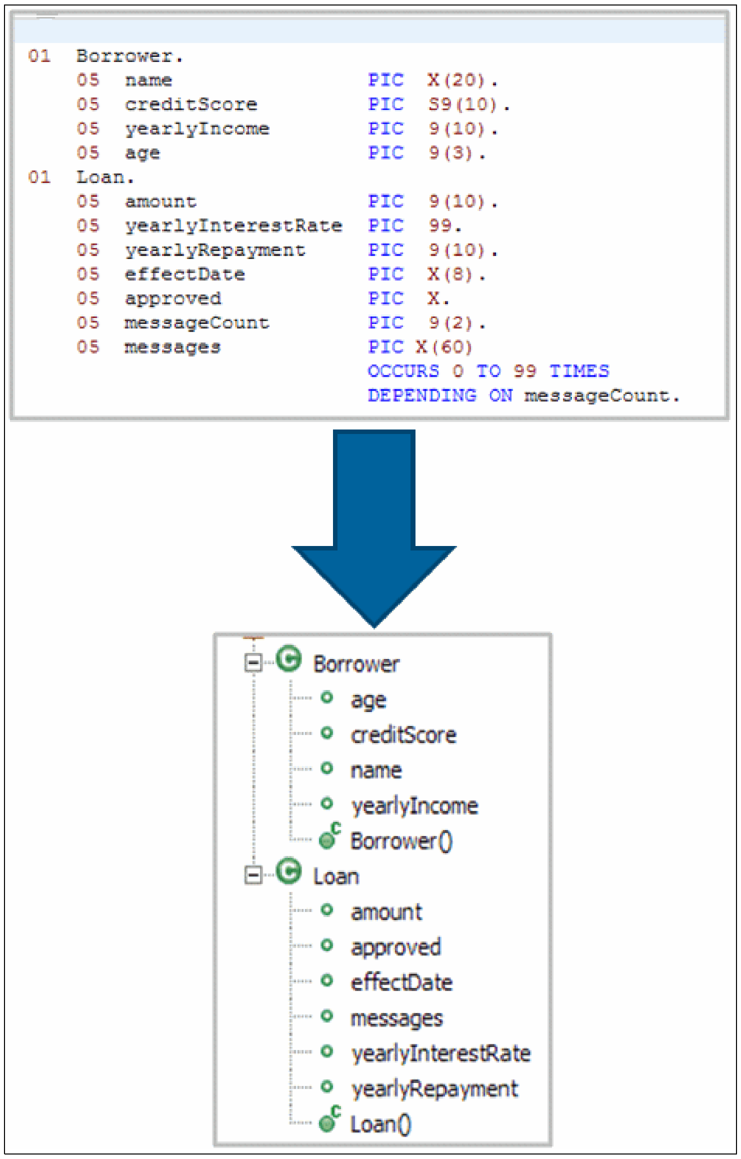
In the next section, the key concepts of the Rule Designer, Decision Center and Decision Server are explained in more detail.

8.2.2 Create decisions using the Rule Designer

Rule Designer enables the creation of a rule project that contains the Execution Object Model (XOM) and the Business Object Model (BOM). It allows a COBOL copybook or PL/I include file to be used as the starting point for these models as shown in Figure 8-4 on page 78. These models are required to allow the authoring of business decisions based on the business data that is currently being used in the COBOL or PL/I applications.

The Rule Designer can also be used to take an existing rule project based on a Java XOM and enable it for COBOL or PL/I execution. This is achieved by generating the COBOL copybook or PL/I include file necessary for the program to execute the existing rule sets.

> **Note:** Existing rule projects that use a XOM generated from an XSD cannot be enabled for COBOL or PL/I execution.

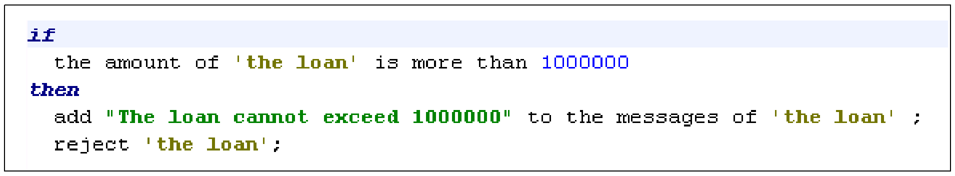


```
01  Borrower.
    05  name                PIC  X(20).
    05  creditScore         PIC  S9(10).
    05  yearlyIncome        PIC  9(10).
    05  age                 PIC  9(3).
01  Loan.
    05  amount              PIC  9(10).
    05  yearlyInterestRate  PIC  99.
    05  yearlyRepayment     PIC  9(10).
    05  effectDate          PIC  X(8).
    05  approved            PIC  X.
    05  messageCount        PIC  9(2).
    05  messages            PIC X(60)
                            OCCURS 0 TO 99 TIMES
                            DEPENDING ON messageCount.
```

```
Borrower
    age
    creditScore
    name
    yearlyIncome
    Borrower()
Loan
    amount
    approved
    effectDate
    messages
    yearlyInterestRate
    yearlyRepayment
    Loan()
```

Figure 8-4 Cobol copybook or PL/I include files are used to create the business and execution models

After the rule project, XOM and BOM are created, the Rule Designer can be used to start authoring the initial decisions. The decisions can be expressed using the Business Action Language (BAL) as shown in Figure 8-5.

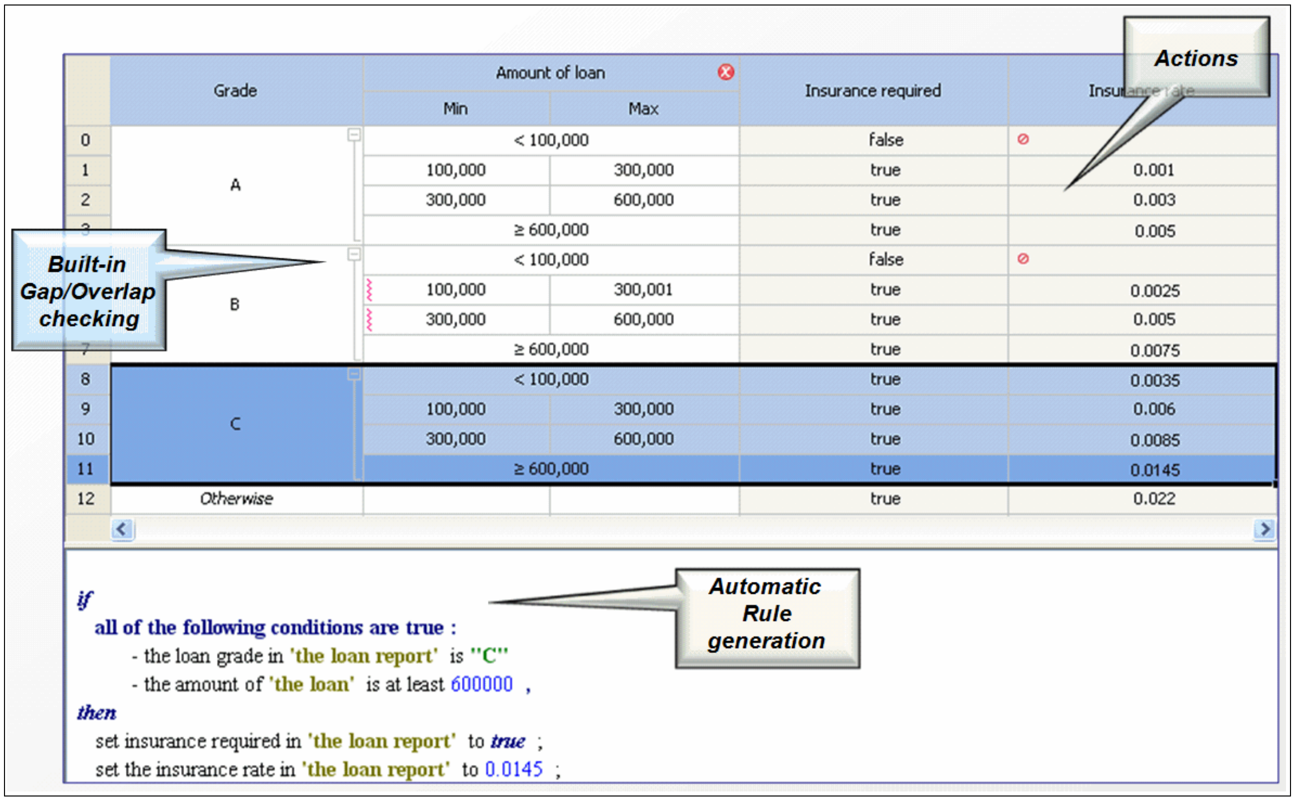


```
if
  the amount of 'the loan' is more than 1000000
then
  add "The loan cannot exceed 1000000" to the messages of 'the loan' ;
  reject 'the loan';
```

Figure 8-5 A business decision expressed in Business Action Language

The BAL provides a simple if-then syntax that is used with a vocabulary to write business decisions. The BAL defines the syntax and provides constructs for expressing conditions and actions, and the vocabulary defines the terms that are used in the business decisions.

Decisions can also be entered using decision tables that provide a graphical way to author decisions, as illustrated in Figure 8-6.



	Grade	Amount of loan		Insurance required	Insurance rate
		Min	Max		
0	A	< 100,000		false	
1		100,000	300,000	true	0.001
2		300,000	600,000	true	0.003
3		≥ 600,000		true	0.005
4	B	< 100,000		false	
5		100,000	300,001	true	0.0025
6		300,000	600,000	true	0.005
7		≥ 600,000		true	0.0075
8	C	< 100,000		true	0.0035
9		100,000	300,000	true	0.006
10		300,000	600,000	true	0.0085
11		≥ 600,000		true	0.0145
12	*Otherwise*			true	0.022

```
if
  all of the following conditions are true :
     - the loan grade in 'the loan report' is "C"
     - the amount of 'the loan' is at least 600000 ,
then
  set insurance required in 'the loan report' to true ;
  set the insurance rate in 'the loan report' to 0.0145 ;
```

Figure 8-6 Decision tables are a graphical way to author decisions

Rule Designer checks for overlaps and gaps in decision table conditions as values are entered in the cells and indicates problems using visual cues. As shown in Figure 8-6, when there is a problem, the column header displays a warning symbol and the affected cells are highlighted.

After the BOM, XOM and business decisions have been created, the rule project can be synchronized with Decision Center for further decision writing, testing, and simulation.

8.2.3 Centrally manage decisions by using the Decision Center

Decision Center is the central hub that coordinates the decision lifecycle across the business and IT parts of the organization. It provides the ability to write decisions, publish reports on existing decisions, validate decisions by using testing and simulation, and deploy decisions to the Decision Server.

Decision Center provides web-based graphical interfaces that allow business users to achieve these tasks with limited dependence on the IT department. The two graphical environments provided are:

Business console The preferred environment for business users to take advantage of social interaction features and change management.

Enterprise console The standard environment for administrators, where deployments can be performed and administrative features such as project security and permission management are available.

After decisions have been edited, tested, and simulated in Decision Center and are ready for execution, they must be deployed to a Decision Server.

8.2.4 Execute decisions by using the Decision Server

Operational Decision Manager for z/OS provides several execution environments in which the Decision Server can run. They are listed here and illustrated in Figure 8-7.

- zRule Execution Server allows COBOL and PL/I applications to execute decisions in a stand-alone server address space or embedded within the client application's address space.
- zRule Execution Server within the CICS TS JVM server allows CICS TS COBOL and PL/I applications to execute decisions in the same CICS TS region or in a remote rule-owning region (ROR).
- Rule Execution Server on WebSphere Application Server for z/OS allows COBOL and PL/I applications to execute decisions via the WebSphere Optimized Local Adapter (Local Adapter) interface.

These execution options are illustrated in Figure 8-7.

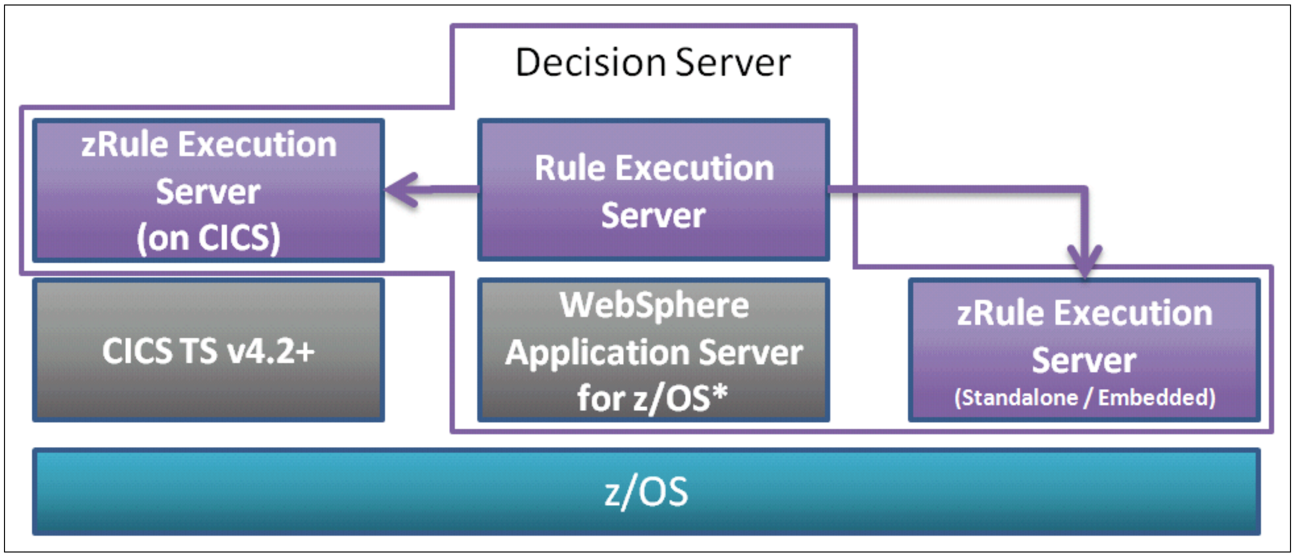

Decision Server
zRule Execution Server (on CICS)
Rule Execution Server
CICS TS v4.2+
WebSphere Application Server for z/OS*
zRule Execution Server (Standalone / Embedded)
z/OS

Figure 8-7 Decision Server on z/OS execution environment option

Note: WebSphere Application Server for z/OS is provided with Operational Decision Manager for z/OS as a limited license product for Decision Center and Decision Server.

The benefits of each execution environment are described more thoroughly in the IBM Redbooks publication titled *Flexible Decision Automation for Your zEnterprise with Business Rules and Events*, SG24-8014:

http://www.redbooks.ibm.com/abstracts/sg248014.html

The execution environment that can be considered as a qualifying workload for execution within IBM CICS Transaction Server for z/OS Value Unit Edition is the zRule Execution Server running within the CICS TS JVM server.

The next section describes how existing COBOL and PL/I programs running in a CICS TS application-owning region (AOR) can execute decisions in a rule-owning region (ROR) running within the JVM server of a IBM CICS Transaction Server for z/OS Value Unit Edition.

8.3 CICS TS rule-owning region architecture

A CICS TS rule-owning region is responsible for hosting Operational Decision Manager for z/OS within its JVM server. It is then possible for other CICS TS regions running on separate LPARs to execute decisions within the ROR. This architecture is illustrated in Figure 8-8.

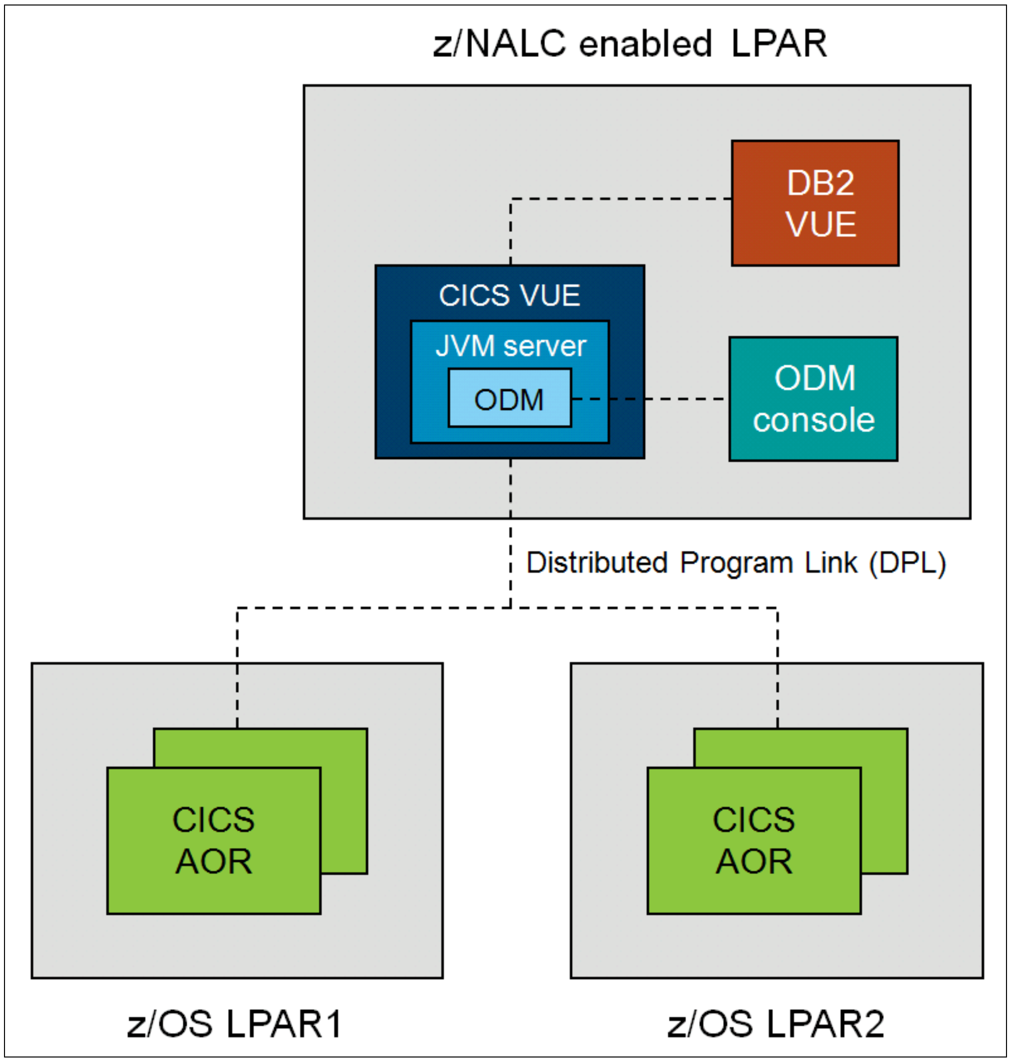


Figure 8-8 CICS TS AORs executing decisions remotely on a zNALC LPAR

Note: The ability to configure a rule-owning region is available in Operational Decision Manager for z/OS version 8.5.1 onwards.

CICS TS AORs can communicate with the ROR using a distributed program link (DPL) or by using IBM CICSPlex® System Manager workload management (WLM). The use of DPL or WLM allows the decision request to be routed dynamically to the ROR. This provides a highly available and work load managed solution when two or more RORs are used.

Then next section details how this architecture also provides a cost-effective way for the COBOL and PL/I applications running in the AORs to execute business decisions using Operational Decision Manager for z/OS running in the ROR.

8.3.1 Cost effectiveness

The cost of Operational Decision Manager for z/OS is based on the size of the LPARs that it is deployed to, regardless of whether Decision Server is running in a zRule Execution Server, WebSphere Application Server for z/OS, or CICS Transaction Server for z/OS. The pricing of Operational Decision Manager for z/OS is not affected by the CICS TS pricing models (VUE or MLC). The products are sold and priced independently.

However, IBM CICS Transaction Server for z/OS Value Unit Edition (VUE) offers a unique way to contain the cost of Operational Decision Manager for z/OS for CICS TS applications that are running in the AORs in two ways:

- The CICS TS rule-owning region allows isolation of the costs of Operational Decision Manager in a single LPAR.

  There is no cost for the Operational Decision Manager for z/OS client libraries running on the AOR. The client libraries provide the API required to execute rules on the ROR.

  Multiple AORs can route decision requests to Operational Decision Manager for z/OS running in the ROR
- zNALC LPARs are separate from z/OS LPARs and are often smaller. Therefore, the cost of Operational Decision Manager for z/OS is reduced.

Operational Decision Manager for z/OS also requires an IBM DB2 database to store the runtime artifacts and to support runtime warehousing features. Another benefit of running Operational Decision Manager for z/OS within a zNALC LPAR is that DB2 for z/OS Value Unit Edition can be used to provide this persistence layer, which further reduces the cost of implementing decision management on z/OS.

8.4 Decision management summary

Making business decisions quickly and correctly is vital for a successful business. This chapter has shown that taking business logic out of application code and storing it centrally in a decision management solution increases the agility, visibility, and maintainability of the business logic.

The recommended method for implementing decision management on z/OS is to use IBM Operational Decision Manager for z/OS, which can support both COBOL and PL/I applications running in batch mode, CICS or IMS. Operational Decision Manager for z/OS can also be deployed to multiple execution environments so that it can be as close to the applications as possible.

Furthermore, Operational Decision Manager for z/OS is Java-based and can run in the CICS TS JVM server. Movement of existing business logic from COBOl or PL/I code to Operational Decision Manager for z/OS is also considered a *net new* workload. Therefore, it is a strong candidate for approval for use within IBM CICS Transaction Server for z/OS Value Unit Edition. This provides a cost-effective way to enable decision management for existing CICS COBOL and PL/I applications. Also, using rule-owning region architecture allows multiple application-owning regions to execute decision requests without the cost of running the Decision Server locally on each AOR.

In the next chapter, a real-world loan application scenario is introduced. The business rules are extracted from the existing COBOL program, coded in rule sets using Rule Designer, and then deployed to Operational Decision Manager for z/OS, running within IBM CICS Transaction Server for z/OS Value Unit Edition. Then, the API calls are made from the COBOL program to execute the business logic in the Decision Server.

9

Implementing decision management in CICS TS

This chapter describes implementing decision management in an IBM CICS integration scenario that is based on a messaging infrastructure that uses IBM WebSphere MQ. We review how business decisions are implemented, deployed, and executed within an IBM CICS Transaction Server for z/OS Value Unit Edition region. The scenario is basd on a loan approval process.

9.1 Objectives

Many banks have CICS based core banking services, such as loan approval processes, that require real-time, nearly instant decisions. The competitive nature of the banking industry means that banks must quickly adapt to changes in the marketplace. This often poses challenges when business decisions are embedded in application code. Any changes take a long time and might cause a financial impact, especially during peak holiday seasons.

IBM Operational Decision Manager for z/OS enables a business to respond to real-time data with intelligent, automated decisions. IT and business users alike can manage the business decision logic that is used by operational systems within an organization. The business decisions are developed and deployed to a Decision Server that runs with the CICS JVM server. CICS based COBOL and PL/I applications can easily invoke the deployed decisions in CICS with the use of simple APIs. Business decisions are changed and deployed without the need for application changes. Using the Operational Decision Manager enables this type of loan processing solution to be highly agile, available, and scalable.

9.1.1 Solution requirements

Table 9-1 shows how the use of IBM Operational Decision Manager server deployed in a CICS JVM server meets the major requirements of the loan processing project.

Table 9-1 Project requirements

Requirement	Solution
Ability to change decision logic quickly and frequently for greater business agility	Modernize applications by incrementally externalizing business rules from COBOL and PL/I applications and moving them to IBM Operational Decision Manager.
Ensure that implementing Operational Decision Manager does not increase costs	IBM Operational Decision Manager server running within a CICS JVM server is eligible to run in CICS VUE regions.
High availability	Because applications can connect to any CICS VUE region in an IBM CICSPlex environment, client applications are not dependent on the availability of a specific CICS region.
High scalability	The use of a CICSPlex enables a scalable solution that takes advantage of the resources across the parallel sysplex. New instances of CICS VUE regions can be easily introduced in the CICSPlex System Manager as business growth dictates.
Workload balancing	The CICSPlex automatically enables full workload balancing.

9.2 Architecture

The key feature of running the Decision Server within the CICS JVM server is its availability across the sysplex. This enables CICS transactions to run in multiple application-owning regions (AORs) and to access the rules that are running in multiple rule-owning regions (RORs). This is illustrated in the high-level architecture for this project shown in Figure 9-1.

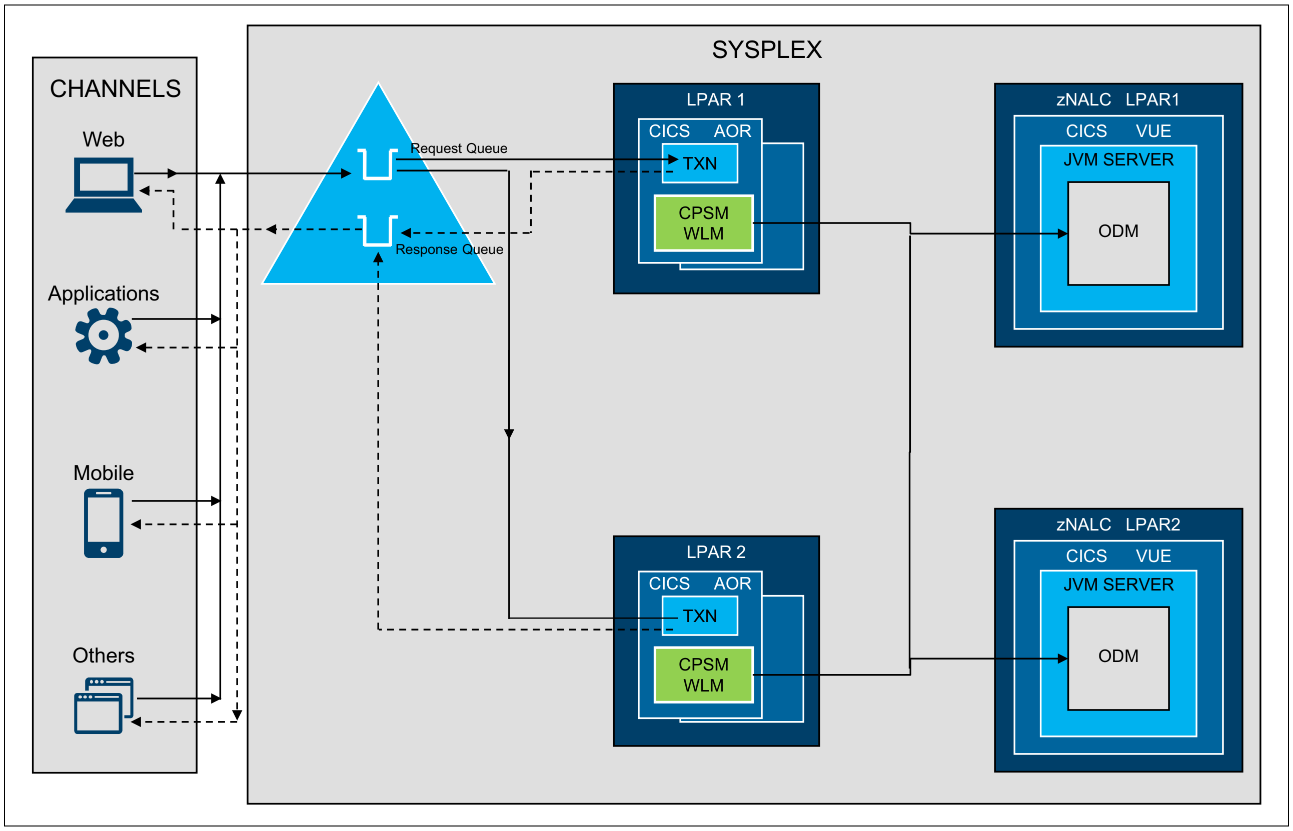


Figure 9-1 IBM Operational Decision Manager in CICS JVM server scenario

In Figure 9-1, you can see how multiple instances of the loan processing application can be running in AORs across the sysplex, with all of them processing messages from the same shared-request queue to maximize throughput by using a request-reply messaging pattern. The scenario consists of the loan processing request arriving from multiple channels in a shared queue. It triggers the loan processing CICS transaction in the AOR.

The loan processing transaction needs to call a decision service to determine loan eligibility. The decision service has been deployed in the Operational Decision Manager Decision Server in the CICS VUE region identified as an ROR. The transaction in the AOR dynamically routes the request to the ROR by using CICSPlex System Manager (SM) Workload Manager (WLM). The use of WLM allows the decision request to be routed dynamically to the ROR. This provides a highly available workload managed solution.

9.3 Implementation

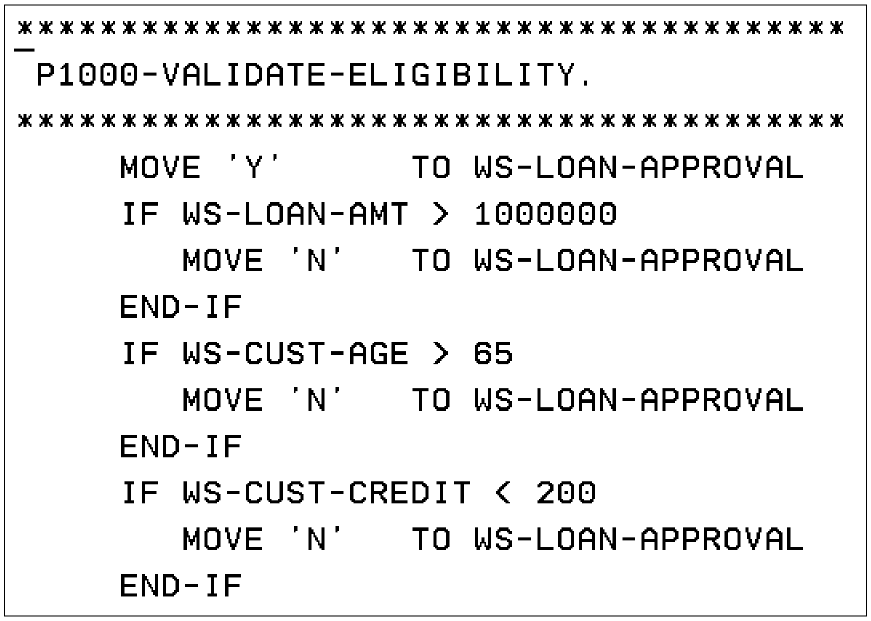
The configuration consists of a set of cloned CICS application-owning regions spread across two LPARs in a parallel sysplex environment. The configuration also consists of a set of cloned CICS rule-owning regions in zNALC LPARs. The CICS AORs share access to a request queue that is held in the coupling facility.

9.3.1 Rule application development

There were multiple phases in the implementation of a rule-based system for the bank:

- Rule discovery, analysis, and design

  The first phase consisted of harvesting , analysis, and design of rules. The eligibility rules are embedded in the IBM client’s CICS COBOL application. The application team and the business analysts worked together in this phase. A sample code snippet of eligibility rules is shown in Figure 9-2.



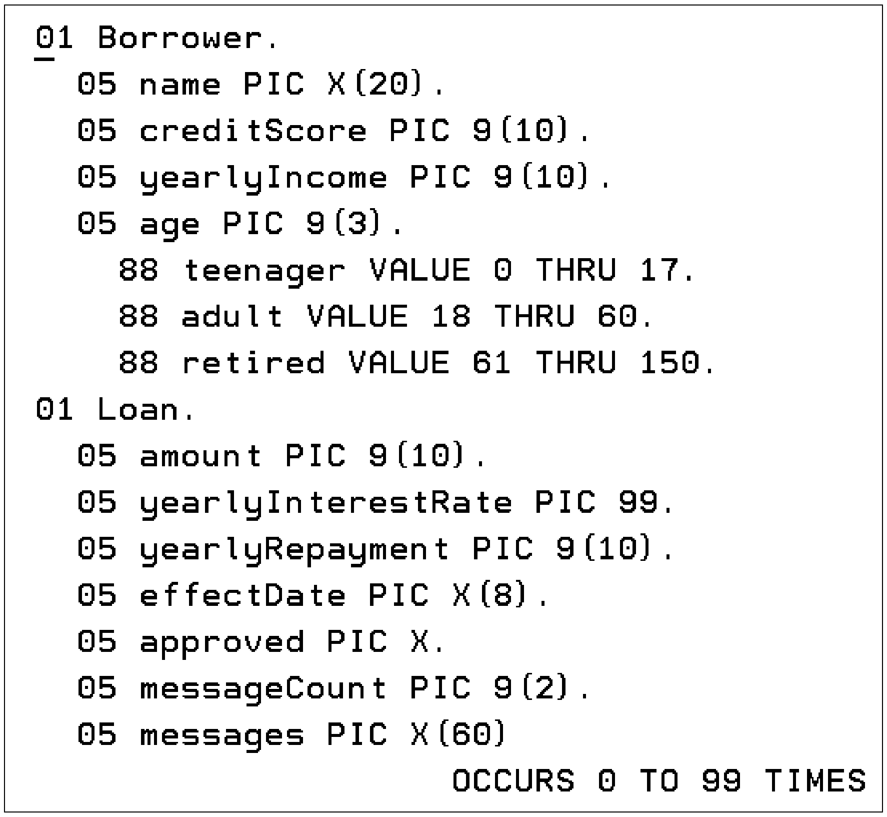
```
****************************************
-
 P1000-VALIDATE-ELIGIBILITY.
****************************************
     MOVE 'Y'      TO WS-LOAN-APPROVAL
     IF WS-LOAN-AMT > 1000000
        MOVE 'N'   TO WS-LOAN-APPROVAL
     END-IF
     IF WS-CUST-AGE > 65
        MOVE 'N'   TO WS-LOAN-APPROVAL
     END-IF
     IF WS-CUST-CREDIT < 200
        MOVE 'N'   TO WS-LOAN-APPROVAL
     END-IF
```

Figure 9-2 Eligibility code

  During the discovery phase, duplicate rules were found embedded in the application code. The business team validated the rules and worked in conjunction with the information technology (IT) team.

  The existing program used a generic copybook for the entire application and was 5 KB long. The loan processing rules use only certain attributes to make a decision, and it is not a good practice to send unwanted data to the rules engine. A new copybook was built for the loan processing rule set. The COBOL copybook layout was designed for parameters to Operational Decision Manager.

The layout of the newly designed COBOL copybook is shown in Figure 9-3.



```
01 Borrower.
   05 name PIC X(20).
   05 creditScore PIC 9(10).
   05 yearlyIncome PIC 9(10).
   05 age PIC 9(3).
      88 teenager VALUE 0 THRU 17.
      88 adult VALUE 18 THRU 60.
      88 retired VALUE 61 THRU 150.
01 Loan.
   05 amount PIC 9(10).
   05 yearlyInterestRate PIC 99.
   05 yearlyRepayment PIC 9(10).
   05 effectDate PIC X(8).
   05 approved PIC X.
   05 messageCount PIC 9(2).
   05 messages PIC X(60)
                          OCCURS 0 TO 99 TIMES
```

Figure 9-3 Rule parameters in the COBOL copybook

- Writing and testing rules

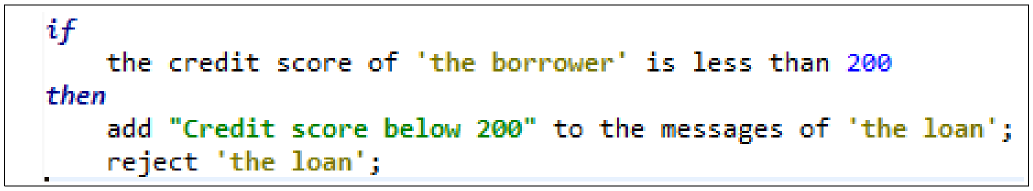
  After the rules are designed, authoring is done through Rules Designer or Decision Center. The rules are then arranged in a rule flow to be exposed as a loan eligibility decision service. Business rules can be authored with any of these:

  – Action rules
  – Decision tables
  – Decision trees

  See “Authoring business rules” in the IBM Knowledge Center for a detailed explanation:

  `http://ibm.co/1zrlExe`

  A sample of the “Minimum Credit Score” rule is shown in Figure 9-4. This is an example of an action rule.



```
if
    the credit score of 'the borrower' is less than 200
then
    add "Credit score below 200" to the messages of 'the loan';
    reject 'the loan';
```

Figure 9-4 Credit score rule

After the rules are developed and packaged, they are arranged in a rule flow, as shown in Figure 9-5. This shows how a loan eligibility decision service is written.

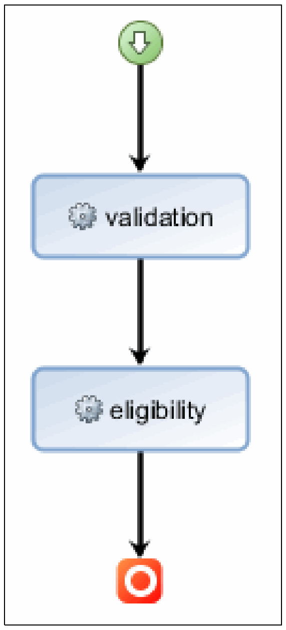
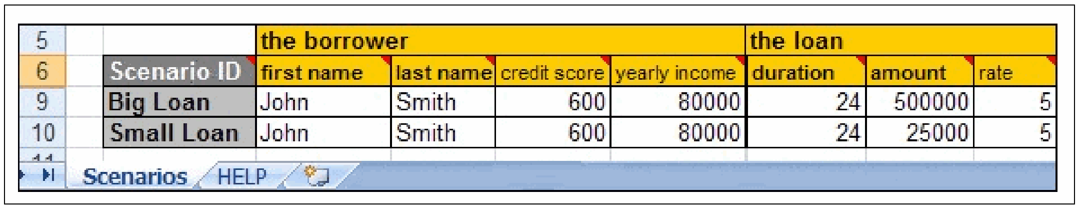


Figure 9-5 Eligibility decision service rule flow

The next step is testing the rule set to be sure that it gives the results that you expect. Decision Validation Services (DVS) are used for testing the accuracy of the rules. The tests are executed in either the Decision Center or Rule Designer, which provides added debugging capabilities. Test scenarios are built with input variables and expected results. A scenario represents the values of the input parameters of a rule set, which include the input data to rule set execution plus any expectations on the execution results that you want to test. The scenarios are built in a Microsoft Excel worksheet, as shown in Figure 9-6.



		the borrower				the loan		
6	Scenario ID	first name	last name	credit score	yearly income	duration	amount	rate
9	Big Loan	John	Smith	600	80000	24	500000	5
10	Small Loan	John	Smith	600	80000	24	25000	5

Figure 9-6 Test scenario parameters

9.3.2 Runtime configuration

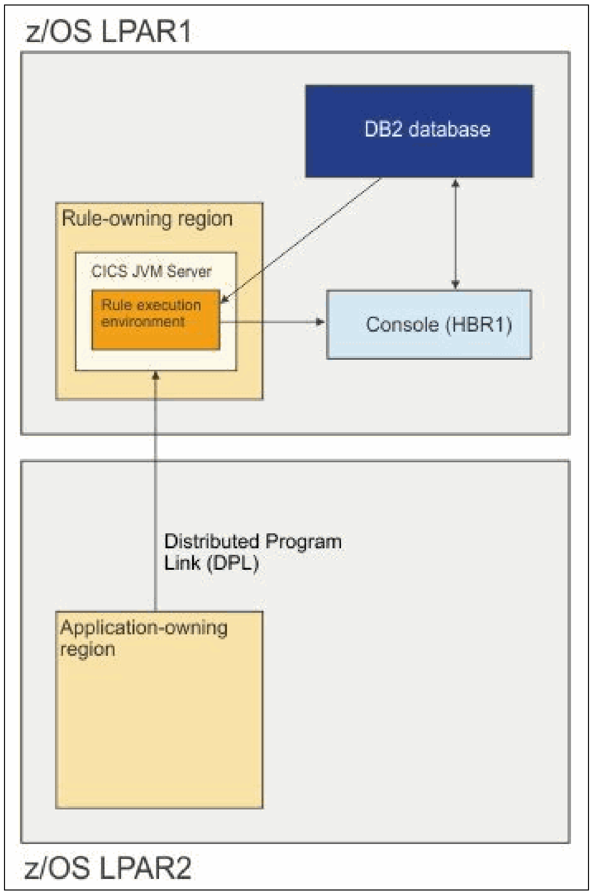
This section describes the runtime configuration.

CICS setup

This topology uses two different types of CICS regions to run rules:

- The rule-owning region (ROR)
- The application-owning region (AOR)

The rule-owning region hosts a zRule Execution Server for z/OS instance that runs locally in a CICS JVM server. The application-owning region uses a CICS distributed program link (DPL) or WLM to run rules in a rule-owning region. The rule-owning region requires a Rule Execution Server console that runs in a separate address space from the CICS region and has a unique subsystem identifier (HBRSSID). A DB2 database provides the persistence layer. The rule-owning region must be run in CICS Transaction Server V4.2 or later. The application-owning region must be on CICS Transaction Server V3.2 or later. Figure 9-7 depicts the topology.


z/OS LPAR1
DB2 database
Rule-owning region
CICS JVM Server
Rule execution environment
Console (HBR1)
Distributed Program Link (DPL)
Application-owning region
z/OS LPAR2
Figure 9-7 CICS AOR and ROR topology

You can find a detailed CICS setup description for ROR and AOR in “Configuring topologies 2 and 3: CICS rule and application-owning regions” in the IBM Knowledge Center:

`http://ibm.co/1rVE62k`

From a CICS TS infrastructure standpoint, the following tasks were completed to configure the topology:

- CICS ROR setup
  - Install Operational Decision Manager for z/OS by using SMP/E.
  - Customize the Operational Decision Manager topology to create the CICS ROR JCL.
  - Run the CICS ROR JCL to create the JVM profile and working paths in HFS or zFS.
  - Enable the CICS started task JCL for DB2 by adding the DB2 libraries in the `STEPLIB` concatenation. The IBM Language Environment® library and `SHBRCICS` Operational Decision Manager library are added to the `DFHRPL` concatenation.
  - Define CICS resources by submitting the `HBRCSD` and `HBRCSDJ` JCL.
  - Edit the CICS System Initialization Table (SIT) parameters for `JVMPROFILEDIR` and `GRPLIST`.
  - Copy the `HBRJVM` profile from the Operational Decision Manager work path location to the `JVMPROFILEDIR` directory for the CICS region.

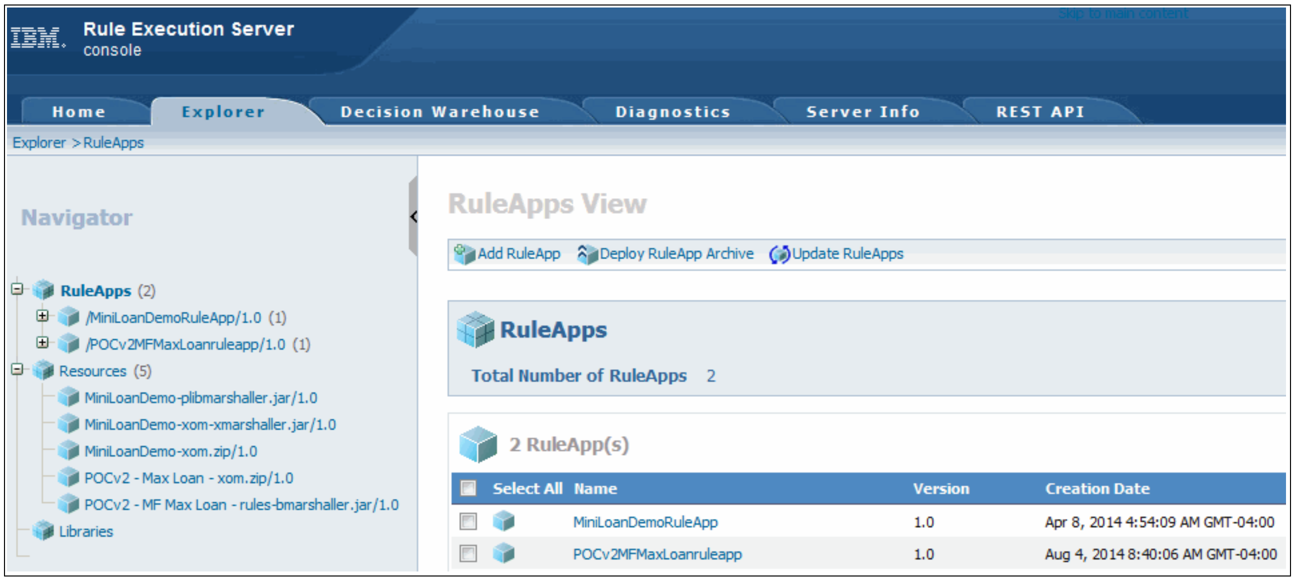
   - Start CICS and issue the HBRC transaction to initialize the Decision Server in CICS.

> **Important:** Ensure that APAR PI07861 is installed on DB2 for z/OS. This resolves an issue where the rule-owning region enters a tight loop when accessing DB2 to load the rule applications.

- CICS AOR setup
   - Customize the Operational Decision Manager topology to create the AOR JCL.
   - Edit the Operational Decision Manager JCL `HBRCSD` so that the `HBRCJVMS` program is defined as a remote server program, and submit the job.
   - Edit the `SHBRPARM(HBRCICSZ)` parm member. Change the *HBRTARGETRES* variable to *RCICSJVM* (for remote execution).
   - Edit the CICS started task JCL to add the SHBRCICS Operational Decision Manager library to the DFHRPL concatenation. Also, add a new HBRENVPR DD card with the data set members `SHBRPARM(HBRCICSZ)` and `SHBRPARM(HBRCMMN)`.
   - Start CICS and issue HBRC transaction.

9.3.3 Deployment and integration

A Rule Execution Server console is a started task on IBM z/OS, and there is only one console for the topology described. The Rule Execution Server console provides a web-based graphical interface that you use for managing and monitoring RuleApps, rule set archives, and Java execution object models (XOMs). It can also be used to monitor execution traces from the Decision Warehouse, test rule set execution, and view server information. Figure 9-8 on page 92 shows a screen capture of the Rule Execution Server console.


IBM. Rule Execution Server
console
Home
Explorer
Decision Warehouse
Diagnostics
Server Info
REST API
Explorer > RuleApps
Navigator
RuleApps (2)
/MiniLoanDemoRuleApp/1.0 (1)
/POCv2MFMaxLoanruleapp/1.0 (1)
Resources (5)
MiniLoanDemo-plibmarshaller.jar/1.0
MiniLoanDemo-xom-xmarshaller.jar/1.0
MiniLoanDemo-xom.zip/1.0
POCv2 - Max Loan - xom.zip/1.0
POCv2 - MF Max Loan - rules-bmarshaller.jar/1.0
Libraries
RuleApps View
Add RuleApp
Deploy RuleApp Archive
Update RuleApps
RuleApps
Total Number of RuleApps 2
2 RuleApp(s)

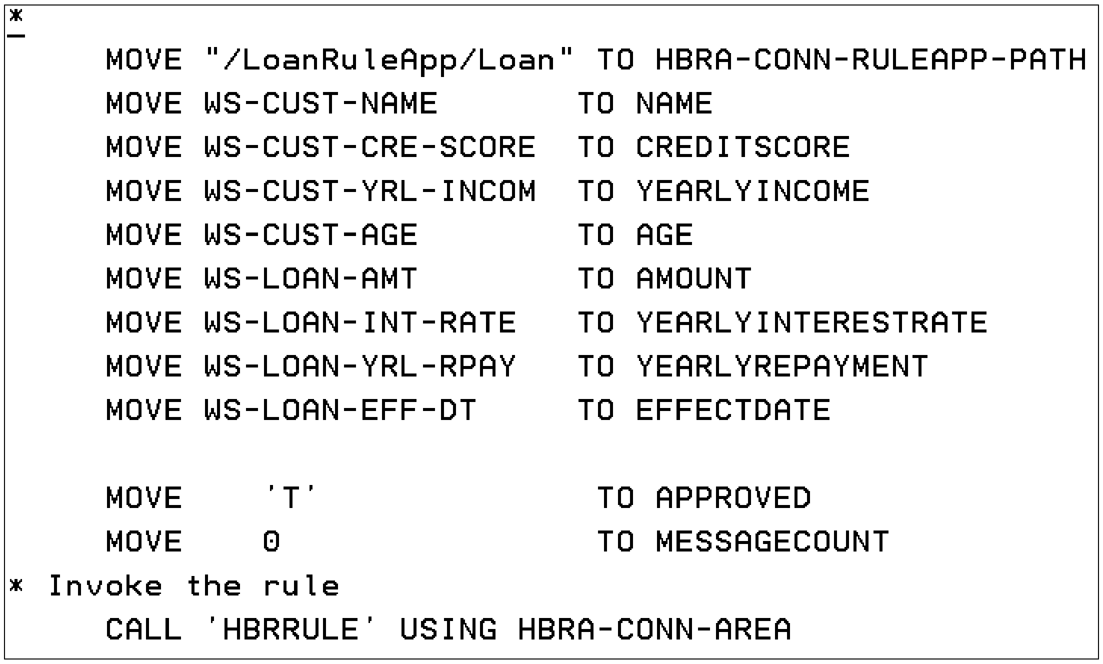
Select All	Name	Version	Creation Date
☐	MiniLoanDemoRuleApp	1.0	Apr 8, 2014 4:54:09 AM GMT-04:00
☐	POCv2MFMaxLoanruleapp	1.0	Aug 4, 2014 8:40:06 AM GMT-04:00

Figure 9-8 Rule Execution Server console

After the RuleApp is packaged, it is deployed to all of the zRES servers through the console. There are many ways to deploy the RuleApp. For additional documentation, see "Overview: Deployment options" in the IBM Operational Decision Manager section of the IBM Knowledge Center:

http://ibm.co/1tVk2r1

The next step is to integrate the rule execution within the CICS COBOL program. The existing code was remediated and an API call was introduced, as shown in Figure 9-9.



```
*
_
    MOVE "/LoanRuleApp/Loan" TO HBRA-CONN-RULEAPP-PATH
    MOVE WS-CUST-NAME       TO NAME
    MOVE WS-CUST-CRE-SCORE  TO CREDITSCORE
    MOVE WS-CUST-YRL-INCOM  TO YEARLYINCOME
    MOVE WS-CUST-AGE        TO AGE
    MOVE WS-LOAN-AMT        TO AMOUNT
    MOVE WS-LOAN-INT-RATE   TO YEARLYINTERESTRATE
    MOVE WS-LOAN-YRL-RPAY   TO YEARLYREPAYMENT
    MOVE WS-LOAN-EFF-DT     TO EFFECTDATE

    MOVE    'T'              TO APPROVED
    MOVE    0                TO MESSAGECOUNT
* Invoke the rule
    CALL 'HBRRULE' USING HBRA-CONN-AREA
```

Figure 9-9 Rule Execution API call

The benefit with the Operational Decision Manager approach is the agility. If the business rules are modified and deployed to the Decision Server, the piece of code remains the same.

Figure 9-10 shows the bank's chosen implementation of the CICS AOR and ROR in a CICSPlex environment.

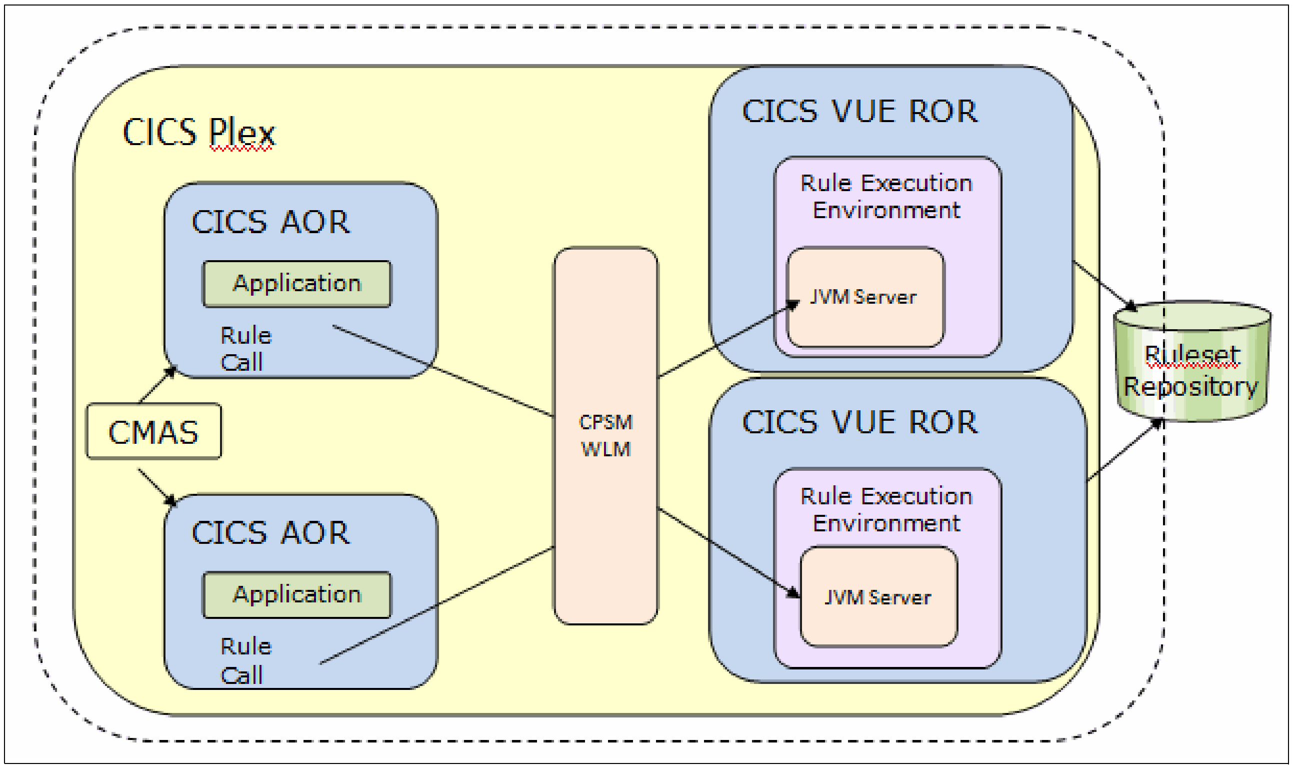


Figure 9-10 CICS and Operational Decision Manager High Availability configuration

The configuration used in Figure 9-8 on page 92 consists of these components:

- Two CICS AORs
- Two CICS RORs

9.4 Solution summary

The use of Operational Decision Manager within a CICS JVM server with a queue-sharing group provides a scalable solution for loan processing and ensures maximum availability for planned and unplanned outages.

Part 5

Modern Batch feature

This part introduces the benefits of using *modern batch* processing, especially when using programs written in Java for batch purposes. It explains how the IBM WebSphere batch environment can be used to schedule and manage batch applications in IBM CICS Transaction Server for z/OS Value Unit Edition.

10

Modern batch workloads

This chapter describes the IBM CICS Transaction Server for z/OS (CICS TS) Feature Pack for Modern Batch, which can be installed in CICS TS V4.2 or later. It enables the WebSphere batch environment to schedule and manage batch applications in CICS.

First, we explain the need for a *modern batch* environment. Then, we review the types of workloads that it is suitable for and the architecture of the solution. The key design considerations when building a batch application to run in CICS TS are highlighted.

This chapter covers the following topics:

10.1 Business pressures on traditional batch processing

Business models have changed. The need now is to have access to data near-real-time data. Therefore, thsoe who still use the traditional batch model are under constant pressure to change to a real-time model. Some of the reasons are described in the following sections.

10.1.1 The “dedicated batch” window is disappearing

There was a time when “batch” and “online” processing were separate from one another. Online processing used to be stopped so that batch processing could have access to the system resources to complete its work.

But those days are behind us. Online processing is becoming a 24x7 operation. This is because client access is increasingly global, with people from different time zones seeking access at all times of the day and night. There might be times when online processing is greatly reduced, but in most cases, it is never stopped altogether. This means that batch and online processing must work at the same time. The window of time available for dedicated batch processing is shrinking. Figure 10-1 shows the batch processing today and the shrinking batch window.

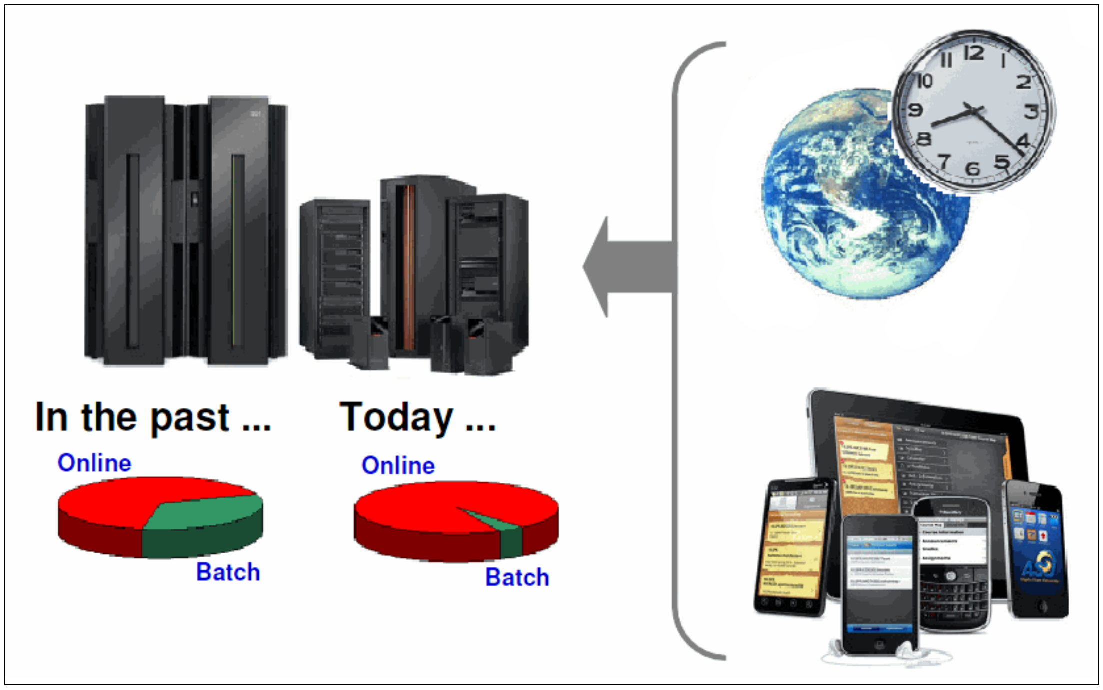


Figure 10-1 Batch processing time available today

The advent of mobile devices means that client access is now even more frequent than before. The transaction work that flows back to information processing systems because of mobile device activity is increasing. Mobile device activity may occur at any time of night or day, so it is truly a 24x7 world for online processing.

But batch processing has not gone away. There are still requirements to do batch work. But it is evident that batch and online work must be processed at the same time and in a manner that implies that both work cooperatively.

10.1.2 The value of shared services

It is not just that the batch window that is shrinking. The cost pressures on maintaining the batch and online transaction processing (OLTP) environments are increasing, too. Having separate infrastructures for online and batch process requires separate computers, separate tools, and separate support staff. Figure 10-2 shows the efficiencies of consolidation of batch and OLTP staff.

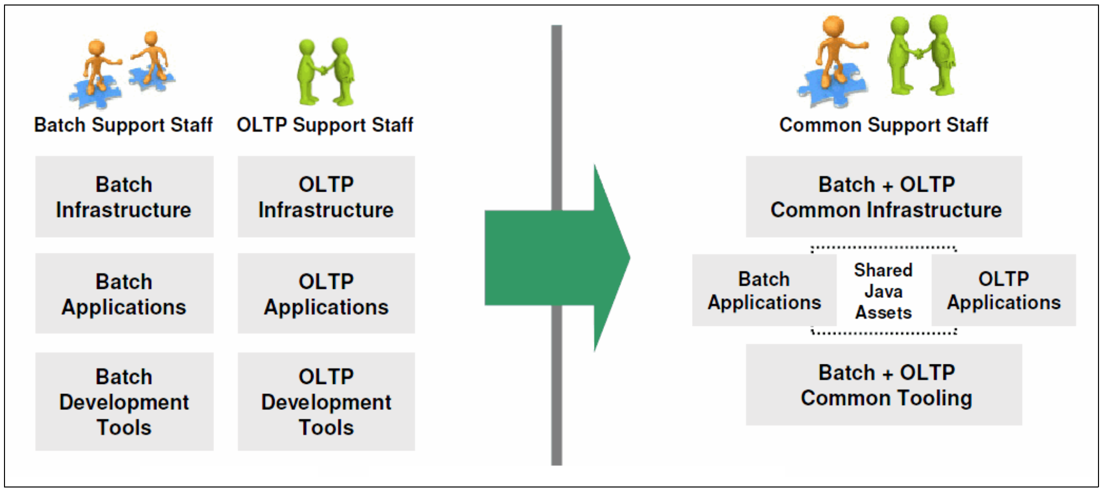


Figure 10-2 Efficiencies through consolidation

The trend is in the direction of convergence. Online and batch processing are both information processing, and, as such, can and should be handled in a cooperative, converged manner. This offers efficiencies in infrastructure, staff, development tools, and, potentially, Java assets that can be shared between OLTP and batch processes.

10.1.3 Java for batch processing

Some readers might ask “Why use Java when COBOL works well?”

Batch assets written in COBOL are still very useful. To the extent that Java is relevant, it is to complement COBOL, not as a total replacement.

However, there are business pressures that make Java an attractive batch solution:

- Skills

  Java skills are simply more plentiful than COBOL skills. Many organizations have very good Java development skills. It makes sense to leverage those Java skills for batch work as well.

- Tools

  Today’s development tools are powerful and quite sophisticated. Acquiring them also represents an investment in technology and an investment in skills. It makes good sense to leverage that investment across multiple information processing disciplines.

- CICS Transaction Server Value Unit Edition (VUE)

  A Java batch workload running in a CICS JVM server is eligible for CICS VUE pricing. This attractive pricing model is a financial motivation for running batches within CICS.

- Specialty engines

  COBOL runs on general processors, but Java runs on specialty engines. Specialty engines (System z Integrated Information Processors, or zIIPs) provide a way to lower overall System z acquisition and licensing costs. Specialty engines are an attractive solution, and leveraging them for batch work is desirable.
- Cooperative processing

  Online processing runtime infrastructures are often designed around Java. For example, CICS Transaction Server for z/OS is a powerful online processing run time. There is an investment in maintaining that online infrastructure. That investment can be leveraged for batch work also.

Therefore, using Java for batch processing is an area of growing interest and is already in use in many large processing operations.

10.1.4 Conflicting needs of CICS applications and z/OS batch applications

It is common for an online CICS application to update resources. For some resources, such as VSAM files, CICS maintains a record and image lock to prevent other applications from making conflicting updates and to be able to restore records in case of a failure. In these cases, the resources need to be opened exclusively by CICS.

A traditional z/OS batch application can be defined as a job written in job control language (JCL) that is submitted to the z/OS job entry subsystem (JES) for execution and does not need user input to complete. For example, it reads all records from a VSAM input file and, for each one, updates a VSAM master file and creates a summary report. However, the master file may be opened for exclusive use by CICS and thus be unavailable to the batch application.

In this scenario, there are several choices:

- Close the master file in CICS and start the batch application, which starts by making a backup of the master file.

  Then, process all of the records, make the updates, delete the backup, and re-open the master file in CICS. If the batch application fails, the backup of the master file is restored, and either the issue is fixed immediately and the batch application is re-started or the issue is fixed later.

  This choice results in a period of time, referred to as the *batch window*, during which the master file and the online applications that use that file are not available.
- Code the batch application to send a request to an online CICS application to make each update to the resources locked by the online application.

  If each request is committed individually, for example, by using the CICS nontransactional External Communications Interface (EXCI), that causes data integrity issues if the batch application fails. This because if the batch application is restarted, some updates will be executed twice. If all requests were committed together, for example, using transactional EXCI, this can result in many records being locked by the online application for an extended time, causing unacceptable delays for other applications.
- For VSAM resources, use record-level sharing (RLS) to maintain record locks for the batch application. However, the batch application is unlikely to maintain its own recoverable logs due to the complexity of writing them. Therefore, if the batch application fails, the records already updated are not restored, and that leads to data integrity issues.

- For VSAM resources, use Data Facility Storage Management Subsystem (DFSMS) transactional VSAM services to lock and log record images before updates. However, unless the application implements its own checkpoints, many records can be locked for an extended period, causing unacceptable delays for other applications.

In addition, as companies provide their services across more locations and time zones and customers require services at times of day to suit them, there is a growing need for online applications to be available continuously, 24x7. Also over time, batch applications are expected to process an increasing amount of data and there is a need to drive down costs. Therefore, in the event of the batch application failing, it is unacceptable to restart it from the beginning. Instead there is a requirement to restart from a frequent checkpoint.

10.2 WebSphere Java batch and batch container services

In this section, we describe the batch environment, WebSphere Java batch, and the batch container framework.

10.2.1 Definition of a batch environment

The *batch environment* is a managed environment for batch applications that are scheduled, process large amounts of information, and may take many hours to complete. The batch environment provides a powerful failover model based on checkpoint and restart scenarios. This is essential to efficiently manage, run, and restart batch applications, in particular when batch application resources are shared with online transaction processing.

The batch environment has two primary components:

- The job scheduler is responsible for determining when and where to dispatch a job, monitoring the job, and reporting back to its caller.
- Endpoints (batch containers) are where the batch application runs. Jobs are dispatched to an endpoint from the job scheduler. The job runs and, upon completion, the job log and return code is provided back to the job scheduler.

10.2.2 CICS functions

CICS is a modern general-purpose transaction processing environment for online applications that start as a result of a request received through a terminal, web service, or message. It typically processes a small amount of information within subseconds. CICS provides the following capabilities:

- Administration, security, and transaction facilities, such as authorization, data integrity, workload management, logging, tracing, debugging, statistics, and monitoring
- API and development tools, such as named counter and XML conversion
- Shared access to resources, such as temporary storage, data tables, IBM DB2 databases, IBM Information Management System (IMS), and Virtual Storage Access Method (VSAM) data sets or files
- Communications, such as web services, WebSphere Optimized Local Adapters, IBM MQ, HTTP, and sockets

10.2.3 WebSphere Java batch

First, it's necessary to understand the high-level architecture of IBM WebSphere Java batch WebSphere Application Server software provides what is known as a "Java EE" (Java Enterprise Edition) runtime. Java EE is an industry-standard specification for an application server that provides a long list of standard application specifications.

Part of the design of this Java EE runtime is the concept of a *container*. Containers are simply runtime functions that provide managed services to the programs that run in the containers. Container-managed services mean tjat the applications can focus on their core business logic and not have to implement common functions. WebSphere Application Server already had web and Enterprise JavaBeans (EJB) containers. The addition of the WebSphere Java batch function adds a third batch container.

Like other containers, the batch container provides function services to the batch applications that run in the container. The CICS Modern Batch Feature pack extends WebSphere Java batch management and execution realm. It allows CICS TS to participate as a WebSphere Java batch endpoint server. Figure 10-3 shows a summary listing of some of those services provided by the batch container of a WebSphere Java batch.

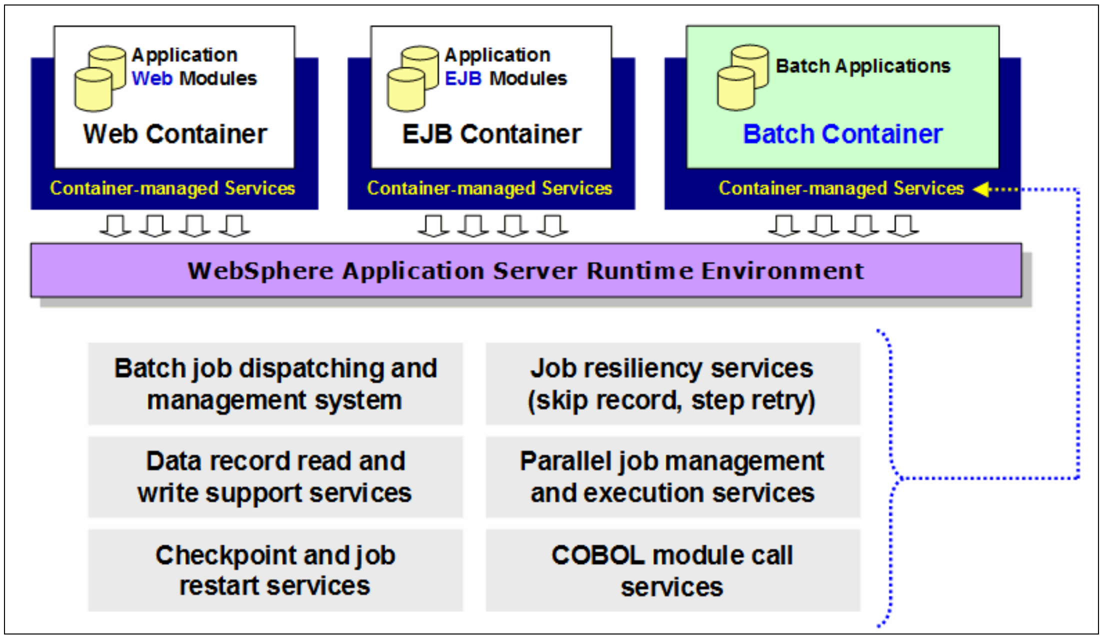

Application
Web Modules
Web Container
Container-managed Services
Application
EJB Modules
EJB Container
Container-managed Services
Batch Applications
Batch Container
Container-managed Services
WebSphere Application Server Runtime Environment
Batch job dispatching and management system
Job resiliency services (skip record, step retry)
Data record read and write support services
Parallel job management and execution services
Checkpoint and job restart services
COBOL module call services

Figure 10-3 WebSphere batch container

The next thing to understand is the job management and execution model provided by WebSphere Java batch. The Java batch separates several key elements of batch processing, as shown in Figure 10-4 on page 103:

- *Job submission:* This is done through a defined interface called the Job Management Console (JMC). It provides a view of the batch environment and allows you to submit and manage jobs.
- *Job description:* The job description is specified in an XML file called *xJCL.* This avoids hardcoding job properties in the application code. The job properties file is used to tell the job submission function what the job is and how to run it. We will describe `xJCL` later in the chapter.
- *Job dispatching:* The job dispatching function signals to the endpoint to begin execution of the batch code named in the job declaration file (xJCL). The job dispatching function

interprets the xJCL, dispatches the job to the endpoint where the batch application resides, and provides ability to stop and restart jobs.

- *Job execution:* The execution of the job takes place in an *endpoint*, which is a batch container where the Java batch application is deployed.
- *Job development and deployment:* Java batch applications implement the batch logic, and they are deployed to the batch containers. The development libraries and tools assist in the creation of the batch applications.

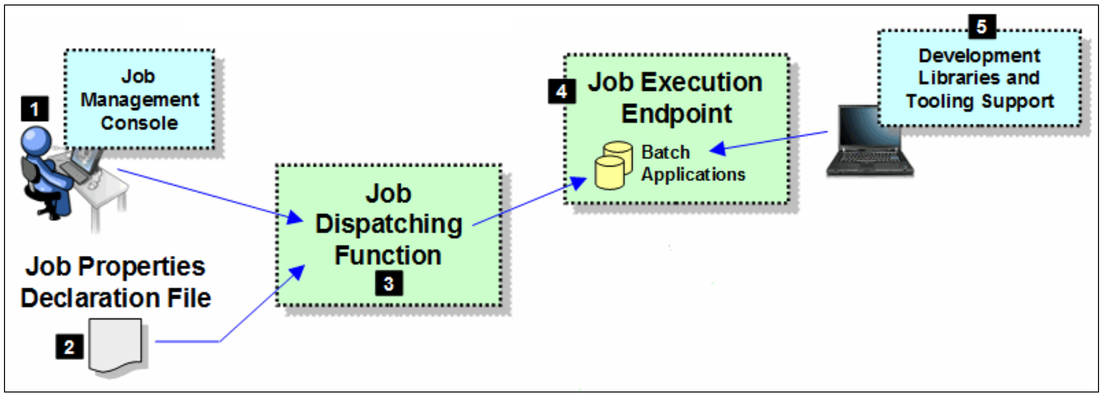


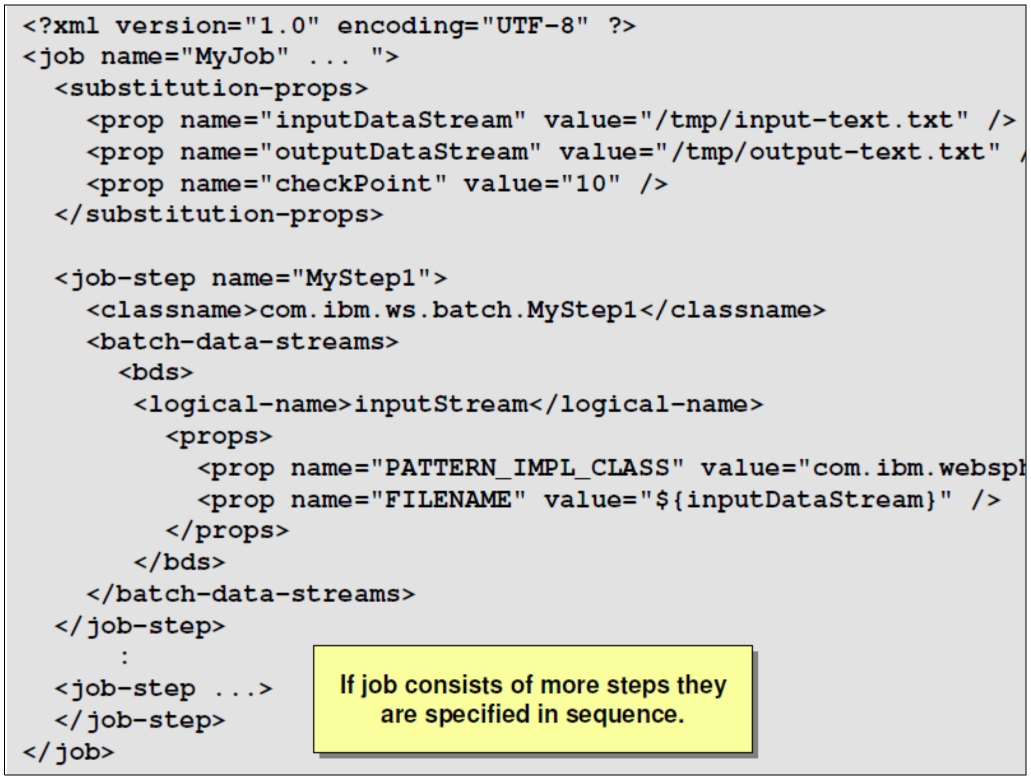
Figure 10-4 Job Management and Execution Model

In the section that follows, we describe the job control language, integration with enterprise schedulers and the batch middleware framework that are supported in CICS TS Feature Pack for Modern Batch.

10.2.4 Job control language

The job control language for modern batch is an XML file called *xJCL*. It describes the Java class files that are used in a batch step and the steps that are included in the batch job. The first thing to understand is the concept of xJCL.

xJCL is a job declaration file. Conceptually, xJCL is just like a normal JCL. It describes the job to be run and the context in which the job is to operate. The difference is that xJCL is written in XML. The xJCL file is used to tell the batch run time that a job invocation is being requested and to provide the runtime understanding of what job to run and to details about the job (like a regular JCL). Figure 10-5 on page 104 shows a trimmed version of an actual xJCL file.



```
<?xml version="1.0" encoding="UTF-8" ?>
<job name="MyJob" ... ">
  <substitution-props>
    <prop name="inputDataStream" value="/tmp/input-text.txt" />
    <prop name="outputDataStream" value="/tmp/output-text.txt" /
    <prop name="checkPoint" value="10" />
  </substitution-props>

  <job-step name="MyStep1">
    <classname>com.ibm.ws.batch.MyStep1</classname>
    <batch-data-streams>
      <bds>
       <logical-name>inputStream</logical-name>
         <props>
           <prop name="PATTERN_IMPL_CLASS" value="com.ibm.websph
           <prop name="FILENAME" value="${inputDataStream}" />
         </props>
       </bds>
    </batch-data-streams>
  </job-step>
       :
  <job-step ...>
  </job-step>
</job>
```

If job consists of more steps they are specified in sequence.

Figure 10-5 xJCL file snippet

10.2.5 Integration with enterprise schedulers

The point of integration on z/OS is still with the enterprise scheduler submitting JCL. Figure 10-6 shows how the integration of enterprise schedulers and the job dispatcher function happens.

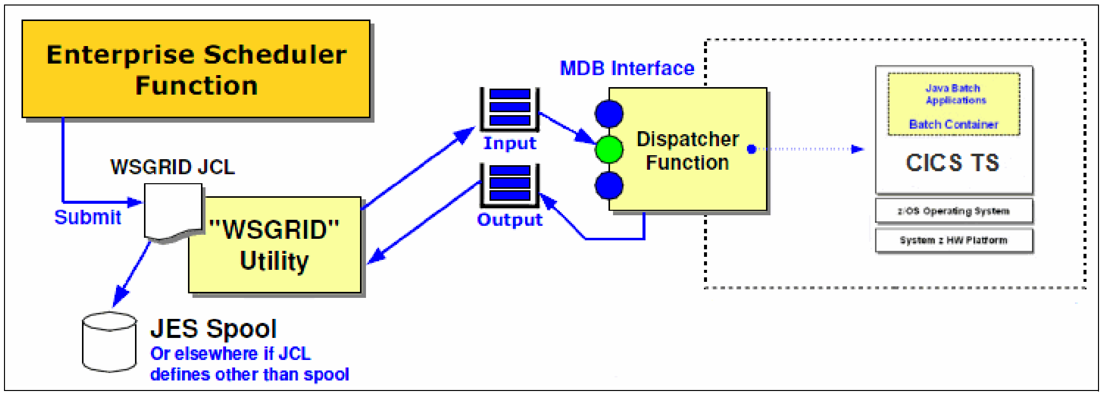


Figure 10-6 Integration with enterprise schedulers

The job dispatcher function is hosted in a WebSphere Application Server and has several interfaces. The one used for this integration is a message-driven bean (MDB) interface. The “glue” between enterprise scheduler JCL submission and the dispatcher is a supplied program that uses messaging (IBM MQ or the service integration bus, or SIBus, of WebSphere Application Server) to submit Java batch jobs to WebSphere Java batches. That “glue” utility is known as WSGRID. The enterprise scheduler sees a batch job as the WSGRID invocation.

10.2.6 Checkpoint and job restart services

Checkpoint commit and rollback is a function of the batch container. This function is abstracted away from the batch job and is handled by the batch container. It relies on the CICS sync point API to do this. These key items are important to note:

- Checkpoint interval (record or time) is specified in the xJCL file.
- As checkpoint intervals are reached, the container commits the records that the checkpoint attained.
- In the event of a failure, the job may be restarted at the last good checkpoint.

Figure 10-7 shows the concepts of checkpoint processing.

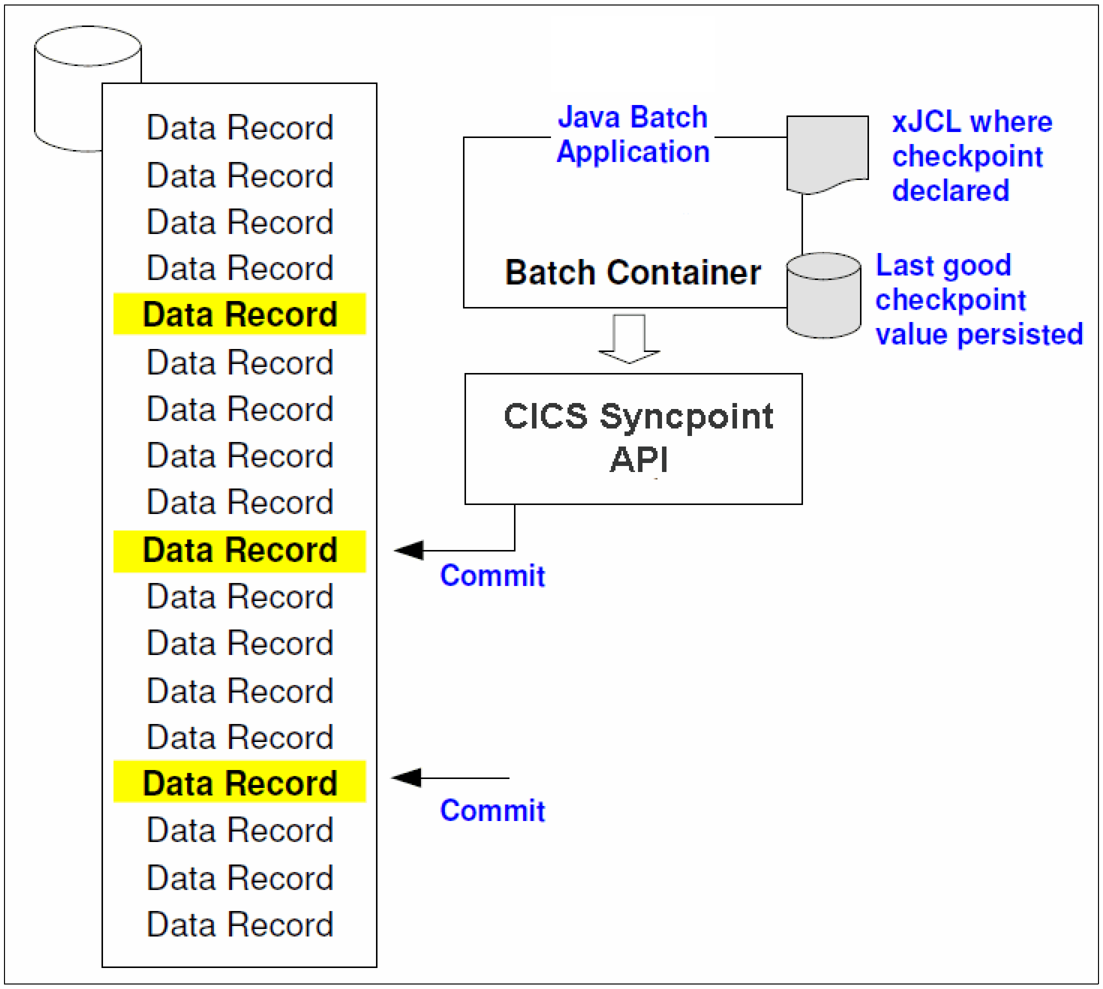


Figure 10-7 Checkpoint processing

10.2.7 Data record read and write support services

The batch container provides *batch data streams* (BDS) for externalizing data access from the job step. This provides a way of abstracting the data read and write logic away from the batch step code.

Several batch data streams are provided:

- Read and write byte data from file
- Read and write text file
- Read from a VSAM key-sequenced data set (KSDS) as input data to the job
- update in a VSAM KSDS data set
- Retrieve data from a database using a JDBC connection
- Write data to a database using a JDBC connection

The Feature Pack for Modern Batch also provides access to the full set of Java class library for CICS (JCICS) APIs.

10.2.8 Job resiliency services

The batch container provides services for the batch job to skip records where a data read or write operation throws an exception. It can also retry job steps for an unhandled exception.

Skip-record processing

This service provides a way of tolerating a data read or write errors so the job can continue. The objective is to provide mechanism to survive the odd data exception rather than stop the job. Figure 10-8 shows skip-record processing.

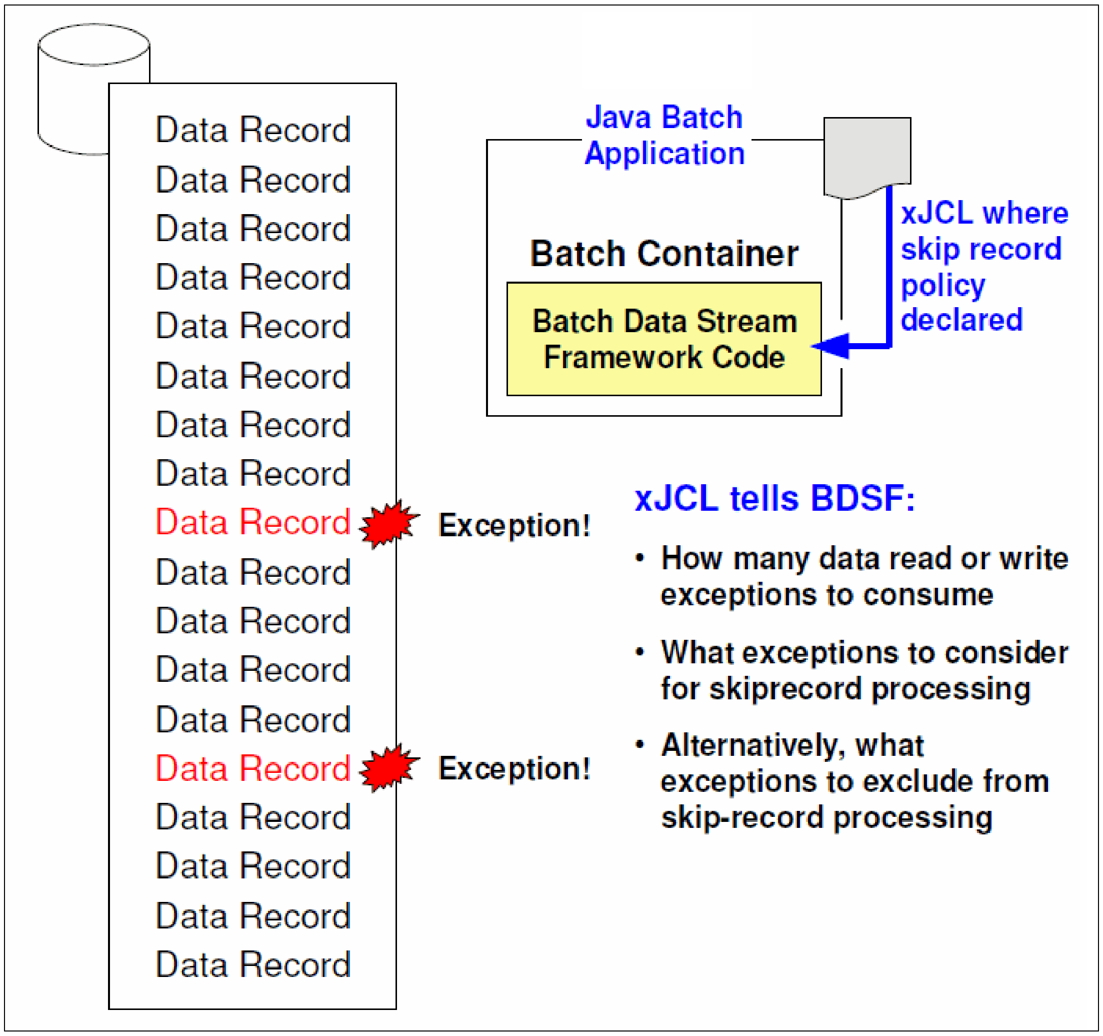


Figure 10-8 Skip-record processing

Retry-step processing

This service provides a way of retrying the job step in the event of an exception. If successful on retry, the job continues and your processing completes. This is at a higher level from skip-record processing. It is at the “invoke batch step” level. This provides a way to retry the step for exceptions. The batch container falls back to the last good checkpoint and restarts from there. Figure 10-9 depicts the retry-step processing. The xJCL provides the following information to the container:

- How many retry steps may be attempted
- What exceptions to consider for retry-step processing
- Alternatively, what exceptions to exclude from retry-step processing

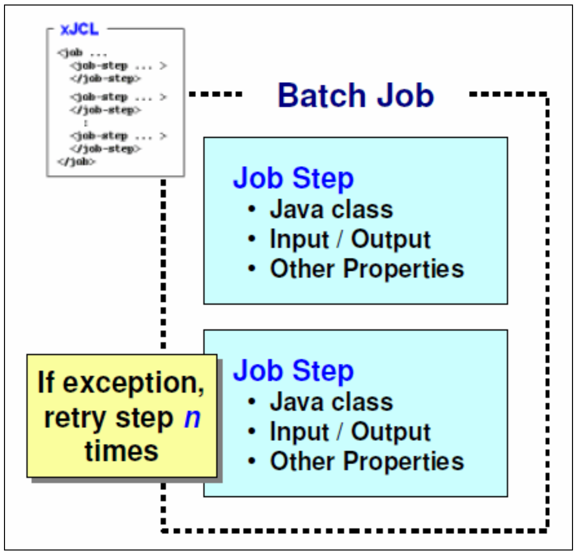


Figure 10-9 Retry-step processing

10.3 Introduction to CICS batch support

In this section, we describe CICS support for modern batches and how batch processing fits within the CICS environment.

10.3.1 CICS support for modern batch

The CICS TS Feature Pack for Modern Batch installs in the CICS region and presents to a WebSphere Java batch dispatcher (dispatcher) as another endpoint to which it can dispatch work. That provides Java batch management for the WebSphere Java batch runtime model, with CICS being an endpoint.

The Feature Pack for Modern Batch function is configured to communicate over the network to the dispatcher, telling the dispatcher about the CICS Feature Pack presence and the Java batch application deployed there. When an xJCL file is submitted to the dispatcher for the Java batch program deployed in the CICS Feature Pack, the dispatcher communicates across the network to invoke and monitor the progress of the batch program. This allows a CICS region to become a job endpoint for a WebSphere Java batch dispatching server, which puts batch logic much closer to the CICS data, as shown in Figure 10-10 on page 108.

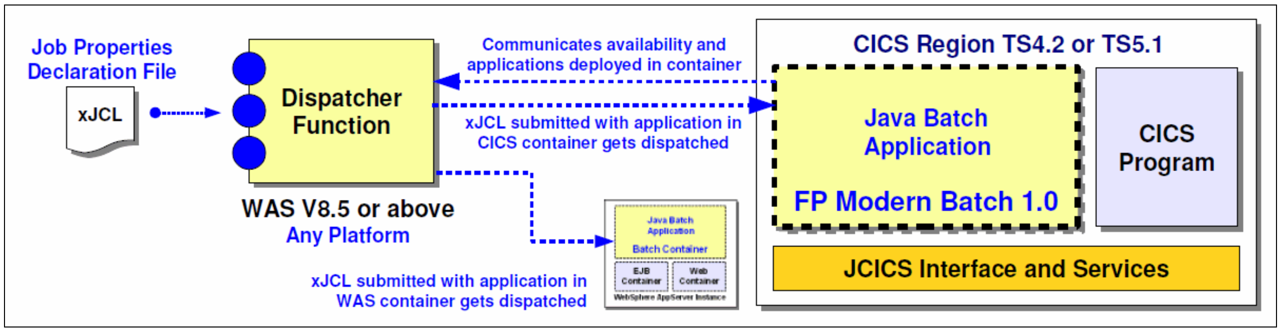

Job Properties Declaration File
xJCL
Dispatcher Function
WAS V8.5 or above
Any Platform
Communicates availability and applications deployed in container
xJCL submitted with application in CICS container gets dispatched
xJCL submitted with application in WAS container gets dispatched
Java Batch Application
Batch Container
EJB Container
Web Container
WebSphere AppServer Instance
CICS Region TS4.2 or TS5.1
Java Batch Application
FP Modern Batch 1.0
CICS Program
JCICS Interface and Services

Figure 10-10 CICS TS Feature Pack for Modern Batch

10.4 Running batch applications in CICS

In this section, we review a solution to resolve the conflicting needs of batch and online applications. It can be used to develop a batch application that uses the batch programming model and runs the application in CICS.

These are the key behavioral aspects of such a batch application:

- The batch application shares access to resources with online applications.
- The batch application creates regular checkpoints to free up transactional resourcest so that online applications are not blocked from completing for excessive amounts of time.

Note: The endpoint provides this checkpoint capability on behalf of the applications.

- If the batch application fails, it can be restarted from its most recent checkpoint.
- Both batch and online applications run concurrently.

The batch application can be divided into job steps that execute in parallel against different subsets of the input data, to shorten the overall elapsed time to process the job.

10.4.1 WebSphere batch environment architecture

The CICS TS Feature Pack for Modern Batch provides an endpoint called the *batch container*, which runs in a Java virtual machine (JVM) in the CICS address space. The job scheduler interacts with the batch container to start, stop, and manage batch applications. These components are required for the batch implementation within CICS TS:

- WebSphere Application Server 8.5 or later
- CICS Transaction Server 4.2 or above
- CICS TS Feature Pack for Modern Batch

Figure 10-11 depicts this architecture and the interaction among the components.

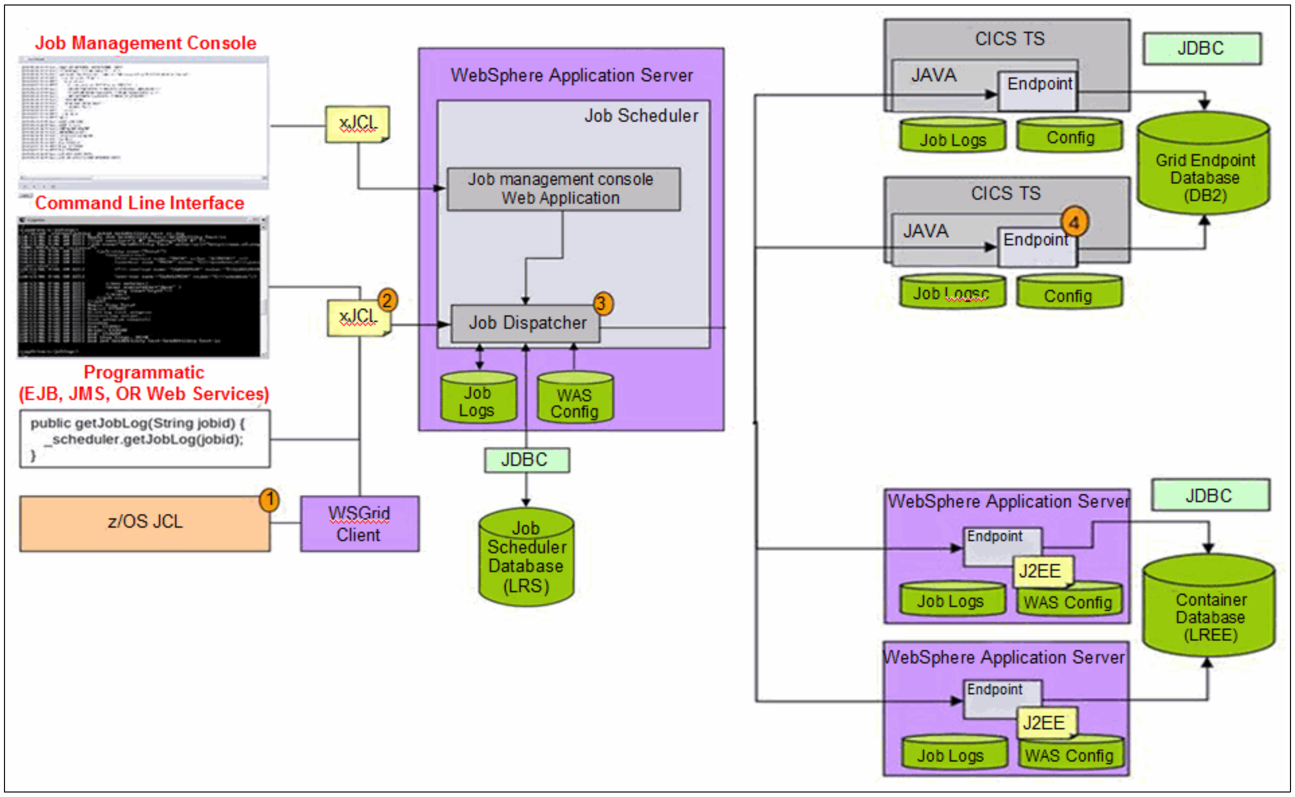

Job Management Console
Command Line Interface
Programmatic
(EJB, JMS, OR Web Services)
public getJobLog(String jobid) {
_scheduler.getJobLog(jobid);
}
z/OS JCL
1
WSGrid Client
xJCL
xJCL
2
WebSphere Application Server
Job Scheduler
Job management console Web Application
Job Dispatcher
3
Job Logs
WAS Config
JDBC
Job Scheduler Database (LRS)
CICS TS
JAVA
Endpoint
Job Logs
Config
CICS TS
JAVA
Endpoint
4
Job Logsc
Config
JDBC
Grid Endpoint Database (DB2)
WebSphere Application Server
Endpoint
J2EE
Job Logs
WAS Config
WebSphere Application Server
Endpoint
J2EE
Job Logs
WAS Config
JDBC
Container Database (LREE)

Figure 10-11   CICS provides an endpoint for the WebSphere batch environment

When started in CICS, the batch container loads a configuration file that details which batch applications it can run, how to connect to the job scheduler, and how the job scheduler can connect to it. The batch container registers with the job scheduler and informs the scheduler of the batch applications it can run. Then, it periodically sends status information to the scheduler, which states that it is still active and available for work.

The starting and processing of a job are detailed by using the numbers in the diagram in Figure 10-11:

1. JCL is submitted on z/OS to request that the batch application start. The JCL runs the WSGRID program and passes the location of an xJCL document to it.
2. The WSGRID program connects to the job scheduler and passes the xJCL location. Alternate interfaces are provided to start a batch application, including a console, command-line interface, and programmatic API.
3. The job scheduler examines the xJCL to establish the name of the batch application to dispatch and uses the information published to it from all endpoints to select which endpoint should run the batch application. The job scheduler chooses a batch container hosted in CICS. It then connects to the batch container, passing the xJCL to it.
4. The batch application runs within the CICS batch container.

When the batch application is running in the batch container in CICS, it can use Java APIs, such as JDBC, or the CICS Java APIs (JCICS) to access CICS resources and services, including VSAM files. It also uses the APIs to call existing CICS programs written in other languages, such as COBOL, C/C++, PL/I, and Assembler.

As the job runs, the batch container takes checkpoints, which enable the batch application to be restarted from the last successful checkpoint in the event of a failure. When the batch application completes, the batch container notifies the job scheduler.

10.5 Reasons to run a batch application in CICS

There are various reasons to run batch applications in CICS:

- When you can reduce costs by taking advantage of the CICS VUE pricing model

  CICS modern batch workload is eligible to run in a CICS VUE region and take advantage of the pricing model. This would substantially reduce the processing costs.

- When there is pressure to reduce or eliminate the batch window

  If CICS is used for online transaction processing and those online applications need to be available for longer each day, it can make sense to run batch jobs in CICS.

- When CICS has opened resources exclusively that are needed for batch processing

  If your batch needs access to resources that are opened exclusively by CICS, it can make sense to run the batch application under the control of CICS.

- When the batch application does not need exclusive resource access

  If the batch application runs at the same time as online applications, updates to the resources being used by the batch job can be made by the online applications. To work in this environment, the batch application needs to be tolerant of these changes.

- When you want to reuse CICS business logic in batch

  Reuse of existing business logic between online applications and batch helps to reduce duplication and can make it quicker and easier to develop new applications. It is likely that there is significant business logic contained within existing CICS online applications. This logic can be invoked using the JCICS equivalent of the EXEC CICS LINK API.

10.6 Benefits of running batch jobs within CICS

The following benefits can result from running batch jobs in CICS:

- Online CICS applications can be available closer to 24 hours a day.

  By running online applications and batch applications in parallel, there is less need to take the CICS managed resources offline.

- Capabilities provided by the batch container simplify application development.

  Capabilities such as checkpointing, recovery to last checkpoint after a failure, logging, and trace are provided by the batch container, removing the need for the application developer to provide these capabilities.

- A common batch programming model is used.

  The batch environment provides a common batch programming model across runtimes and platforms. Therefore, any developers who are skilled in batch environment application development should be able to write a batch application to run in CICS.

- People with Java skills are readily available.

  Java is a well-known, popular, modern language. Batch job steps and batch data streams are written in Java.

- Modern batch within CICS is eligible for Value Unit Edition pricing.

  CICS modern batch workload is eligible to run in a CICS VUE region and take advantage of the pricing model. This substantially reduces the processing costs.
- Java functions are included.

  Functions such as email, PDF file generation, and XML processing are readily available for Java. This can make it easy to develop batch functions that might be more challenging to implement in languages such as COBOL or PL/I.

10.7 Implications of running batches in CICS

Running a batch job under the control of CICS means that the job has different behavioral characteristics. These are some of the implications of running a batch job in CICS.

- Batch jobs might take longer to run.

  Traditional z/OS batch jobs are typically optimized to run as quickly as possible. When a batch job is moved to CICS, it might have to share resources with online applications. These resources can be CPU resources, files, or databases. Therefore, it is possible that the batch job might take longer to complete. At the same time, with the removal of the batch window, it might be that the batch job can start earlier or complete later. For batch jobs with hard time deadlines, investigation is required to understand whether the job can complete in the time required.
- Implications regarding online application performance need to be considered.

  It is important that the batch processing does not negatively affect the performance of online applications. A user who is invoking a CICS transaction might expect a response time of tenths of a second. If the batch application locks too many resources at a time, it could affect the performance of the online applications. The checkpoint interval of the batch job can be adjusted to change the number of updates made within a checkpoint. If the batch job consumes too much of the CPU resources, this can also affect the performance of the online applications.
- Data being updated by batch processing can be changed by online applications.

  Batch applications that rely on a set of records to remain consistent may not work when run in parallel with online applications, because the online applications can update records that the batch job has read or is about to read. This needs to be considered on a per application basis to determine whether it is likely to be an issue.
- Traditional batch jobs need factoring.

  If you plan to move an existing batch job to CICS, the job needs refactoring to fit into the batch programming model.

10.8 Summary

Batch processing has proved to be an efficient, manageable and reliable method for bulk processing of updates to data. As businesses expand, the volume of data to be processed also expands. At the same time, the growth of online transactional workloads suggests that a new paradigm is needed to enable the two processing styles to work better together. The CICS batch container provides this new capability.

11

Modern batch use scenario

This chapter describes an IBM CICS Transaction Server (TS) modern batch scenario where a Java batch application is developed to reduce the batch window. The Java batch accesses data in VSAM or IBM DB2 without the need to stop the online transactions that access to the same data. The Java batch application may come from a new requirement or may be rewritten from existing business logic. We review the architecture and the implementation of the modern batch Java solution.

11.1 Java batch approaches

There are two approaches to using Java batch processing:

- Use Java batch for new applications.

  In this approach, you can leave existing batch processes as they are, but engineer new requirements in the Java batch model. If a batch application has complex algorithms and huge calculations, needs to access to the data controlled by CICS TS where the data access is either in share mode, or holds the exclusive lock or a short time, the application is a good candidate for modern batch processing.

- Re-engineer existing batch processes.

  Re-engineering is not an easy thing to do, but IBM provides tools to help Java interact with structured data. For more information, see Chapter 8 in the IBM Redbooks publication titled *IBM CICS and the JVM server: Developing and Deploying Java Applications*, SG24-8038:

  `http://www.redbooks.ibm.com/abstracts/sg248038.html?Open`

11.2 Architecture

In this section, we introduce the architecture to use modern batch processing.

11.2.1 Workflow

Figure 11-1 on page 115 shows the architecture of how a Java batch works with CICS TS. At the front, enterprise schedulers such as IBM Tivoli Workload Scheduler can be used to schedule a batch request from z/OS JCL. The JCL is submitted to WSGRID client that is provided by WebSphere Application Server and is used to interact with Java batches and to integrate the batch system with the enterprise schedules. The WSGRID client reads the job properties from XML Job Control Language (xJCL) and then sends the request to the job scheduler in WebSphere Application Server. The job scheduler then schedules the job to endpoints (batch containers) in CICS TS.

To allow a CICS TS region to become a job endpoint for a WebSphere Java batch dispatching server, CICS TS TS Feature Pack for Modern Batch V1.0 must be installed. After a Java batch application is deployed in a CICS TS batch container, CICS TS communicates with the WebSphere Application Server dispatcher function host and port. That lets the dispatcher know that the endpoint is present and what batch application is deployed.

The batch container provides *batch data streams* (BDS) as a means of externalizing data access from the job step. For more information, see “Batch data streams provided by CICS” in the IBM Knowledge Center:

`http://ibm.co/1vmpKCw`

In addition, the CICS Feature Pack for Modern Batch provides access to the full set of JCICS APIs. For more information about JCICS services and examples, see “JCICS API services and examples” in the IBM Knowledge Center:

`http://ibm.co/1I6xvXw`

Now, let’s focus more on how to access to VSAM files, because VSAM is the most common data source for batch processing. CICS TS JCICS services support reading and updating a

VSAM file but do not support opening or closing a file. So, if the file is not owned by the Java batch region but it owned by another CICS TS region for online transactions, you have two solutions:

1. *Function ship* file operations to a file-owned region (FOR)

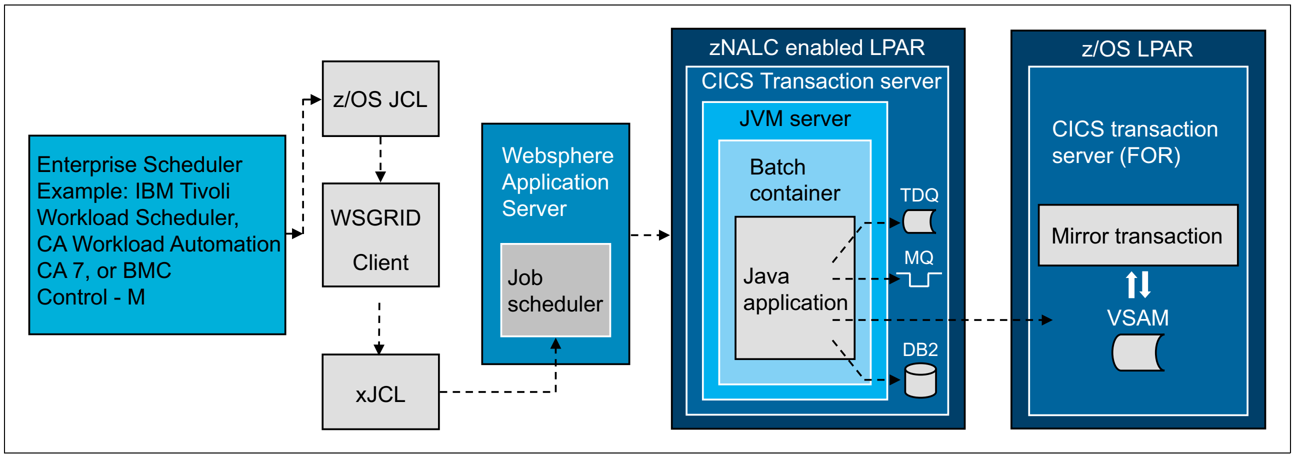
   This solution is shown in Figure 11-1 on page 115. Here, we route all of the file work by function shipping to a FOR running in a normal z/OS LPAR to complete the file read and update.

   In this method, the Java workload is in a zNALC-enabled LPAR, and the workload for file operations is included in a normal z/OS LPAR. You do not need to care about file open and close, and the best thing is that modern batches can run concurrently with online transactions.

2. Use *record-level sharing* (RLS) for VSAM file access

   RLS is another solution that enables modern batch to work concurrently with online transactions and thus reduces the batch window. RLS is a VSAM data set access mode introduced in DFSMS and supported by CICS. RLS enables VSAM data to be shared, with full update capability, between many applications running in many CICS regions. With RLS, CICS regions that share VSAM data sets can reside in one or more IBM MVS™ images within an MVS sysplex.

FOR and RLS are preferred solutions for Java modern batch processing. You can select a solution based on the existing VSAM access method used by your online transactions.



Enterprise Scheduler Example: IBM Tivoli Workload Scheduler, CA Workload Automation CA 7, or BMC Control - M
z/OS JCL
WSGRID Client
xJCL
Websphere Application Server
Job scheduler
zNALC enabled LPAR
CICS Transaction server
JVM server
Batch container
Java application
TDQ
MQ
DB2
z/OS LPAR
CICS transaction server (FOR)
Mirror transaction
VSAM

Figure 11-1 Architecture of a pure Java batch solution

11.2.2 High availability consideration

To achieve the high availability, you can set up two or even more batch containers for the same batch application. The batch containers can be deployed in the same CICS region or in separate CICS regions in the same LPAR, or even in CICS regions in separate LPARs. All of the batch containers send the heartbeat information to the job scheduler at regular intervals. So, from the view of the job scheduler, all of these batch contents are candidates to run the same batch applications. After one container fails, the job scheduler cannot hear the heartbeat from that container, so all of the requests go to the live ones.

Even if you select only one batch container, CICS modern batch processing can achieve better availability than the traditional batch processing by using checkpoints. In the traditional batch, you usually store a backup copy of the master file before running the batch, restore the file after a failure, and then rerun the batch. In the modern batch, the batch container creates checkpoints. The batch application can be restarted from the last successful checkpoint in the event of a failure. This reduces the time to recover from a failure, so it improves availability.

11.2.3 Security consideration

CICS Feature Pack for Modern Batch supports SSL client authentication to secure the communication between the batch container and the job scheduler. You can turn on the security setting in the `batchcontainer-config.xml` file. For more information, see "Configuring the batch container" in the IBM Knowledge Center:

http://ibm.co/1FRNSDl

After the SSL handshake, the job scheduler can schedule the job to the batch containers in CICS. The batch containers use TCPIPService to listen to the requests from the job scheduler. Then, URIMAP is used to route the request to the batch applications. In the URIMPAP definition, you can define the transaction ID and the user ID to run the batch applications. The default transaction is CBCR. Make sure that the transaction is running under an ID that has enough authority to access the resource. As online transactions, the security of the resource is controlled by External Security Manager, which in most cases is IBM RACF®.

11.3 Implementation

In this section, we describe basic steps to develop a Java batch application, deploy it in CICS TS, and schedule it. The IBM Redbooks publication titled *New Ways of Running Batch Applications on z/OS: Volume 1 CICS Transaction Server*, SG24-7991, includes more detailed information about how to use IBM Rational Application Developer to develop a Java batch application and to schedule it:

http://www.redbooks.ibm.com/abstracts/sg247991.html?Open

To use the CICS Feature Pack for Modern Batch, you must have the correct versions of software and service installed. Your environment must comply with the following prerequisites:

- CICS Transaction Server for z/OS, Version 4.2 or later
- CICS Transaction Server for z/OS, Version 4.2 requires APAR PM82511
- CICS Transaction Server for z/OS, Version 5.1 requires APAR PM82519
- IBM z/OS Version 1 Release 12 or later
- IBM DB2 Version 9.1 or later
- IBM WebSphere Application Server Version 8.5 for z/OS or later
  or WebSphere Application Server Network Deployment 8.5 or later

11.3.1 Install and configure CICS TS TS Feature Pack for Modern Batch

CICS TS TS Feature Pack for Modern Batch is installed by using the **`SMP/E RECEIVE`**, **`APPLY`**, and **`ACCEPT`** commands. The program number for CICS TS Feature Pack for Modern Batch is 5655-Y50, and the FMID is HCIF51B. You can use the sample jobs that are provided to perform part or all of the installation tasks, or you can use the SMP/E dialogs to perform the SMP/E installation steps.

For details about the installation procedure, see the Pubilcation Information web page for *CICS Transaction Server Feature Pack for Modern Batch V1.0 Program Directory*, GI13-3324:

http://ibm.co/1FMqucg

Detailed steps are described in “Configuring the CICS Feature Pack for Modern Batch” in the IBM Knowledge Center:

http://ibm.co/15OkVxe

11.3.2 Developing a batch application

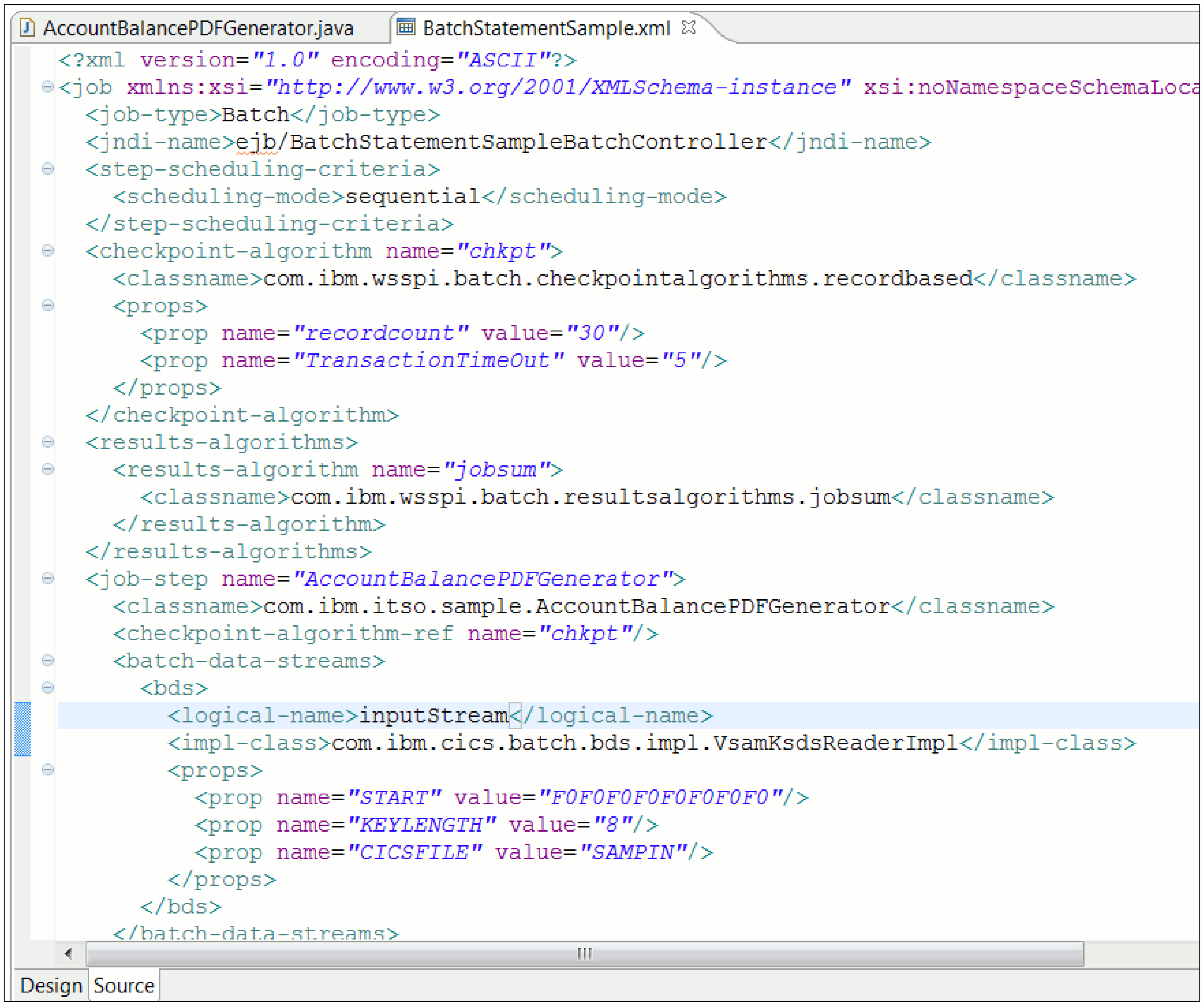
You can develop applications with IBM Rational Application Developer or the CICS Explorer SDK. Rational Application Developer is a preferred IDE to develop the batch application. But if you do not have Rational Application Developer, you can use the Eclipse platform with CICS TS Explore SDK at no charge.

The two most important things that you need to develop are XML Job Control Language (xJCL) and Java class to implement the job step logic.

If you have Rational Application Developer, you can set up a batch project to easily generate xJCL and a Java class template to implement your batch job steps, using a GUI.

Figure 11-2 shows an example of the generated xJCL from Rational Application Developer. The xJCL describes how to run a batch job and defines the input and output streams to that batch job. In the xJCL, you can define batch job steps, checkpoint algorithms, results algorithms, batch data streams (BDS) and batch job return codes. For more information, see “The batch programming model” in the IBM Knowledge Center:

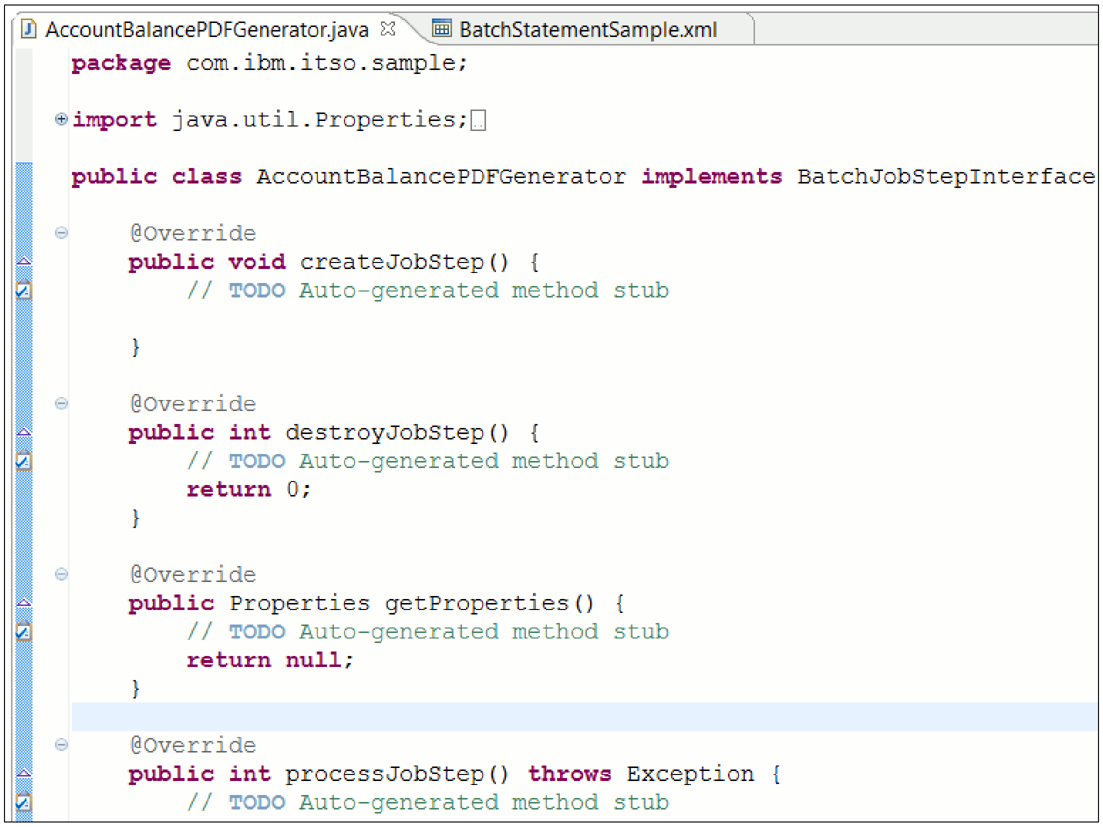
http://ibm.co/1rWywXM



```
<?xml version="1.0" encoding="ASCII"?>
<job xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLoca
  <job-type>Batch</job-type>
  <jndi-name>ejb/BatchStatementSampleBatchController</jndi-name>
  <step-scheduling-criteria>
    <scheduling-mode>sequential</scheduling-mode>
  </step-scheduling-criteria>
  <checkpoint-algorithm name="chkpt">
    <classname>com.ibm.wsspi.batch.checkpointalgorithms.recordbased</classname>
    <props>
      <prop name="recordcount" value="30"/>
      <prop name="TransactionTimeOut" value="5"/>
    </props>
  </checkpoint-algorithm>
  <results-algorithms>
    <results-algorithm name="jobsum">
      <classname>com.ibm.wsspi.batch.resultsalgorithms.jobsum</classname>
    </results-algorithm>
  </results-algorithms>
  <job-step name="AccountBalancePDFGenerator">
    <classname>com.ibm.itso.sample.AccountBalancePDFGenerator</classname>
    <checkpoint-algorithm-ref name="chkpt"/>
    <batch-data-streams>
      <bds>
        <logical-name>inputStream</logical-name>
        <impl-class>com.ibm.cics.batch.bds.impl.VsamKsdsReaderImpl</impl-class>
        <props>
          <prop name="START" value="F0F0F0F0F0F0F0F0"/>
          <prop name="KEYLENGTH" value="8"/>
          <prop name="CICSFILE" value="SAMPIN"/>
        </props>
      </bds>
    </batch-data-streams>
```

Figure 11-2 Extract of the generated xJCL

Figure 11-3 shows an extract of the generated Java class for the Batch Job Step. This class is where the code for the job step will be added.



```
package com.ibm.itso.sample;

import java.util.Properties;

public class AccountBalancePDFGenerator implements BatchJobStepInterface

    @Override
    public void createJobStep() {
        // TODO Auto-generated method stub

    }

    @Override
    public int destroyJobStep() {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public Properties getProperties() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public int processJobStep() throws Exception {
        // TODO Auto-generated method stub
```

Figure 11-3 Extract of the generated Batch Job Step implementation class

After you get the generated Batch Job Step implementation class, you can add your specific business logic there. Batch steps are implemented as plain old Java object (POJO) classes that implement the interface, `com.ibm.websphere.batch.BatchJobStepInterfance`. Batch job steps are performed sequentially. Callback methods in the `BatchJobStepInterface` allow the grid endpoints to run batch steps when they run batch jobs. For more information, see “Batch job steps” in the IBM Knowledge Center:

`http://ibm.co/1ykJicK`

If you do not have Rational Application Developer, you need to develop the xJCL and the job step Java classes manually. Create a Java project to, first, add additional JARs in the build path and then to write a Java class for each job step and write a xJCL file to describe how to run the job. For detailed steps, see “Developing the sample application with CICS Explorer SDK” in the IBM Knowledge Center:

`http://ibm.co/1Aperyx`

11.3.3 Deploying the batch application in CICS

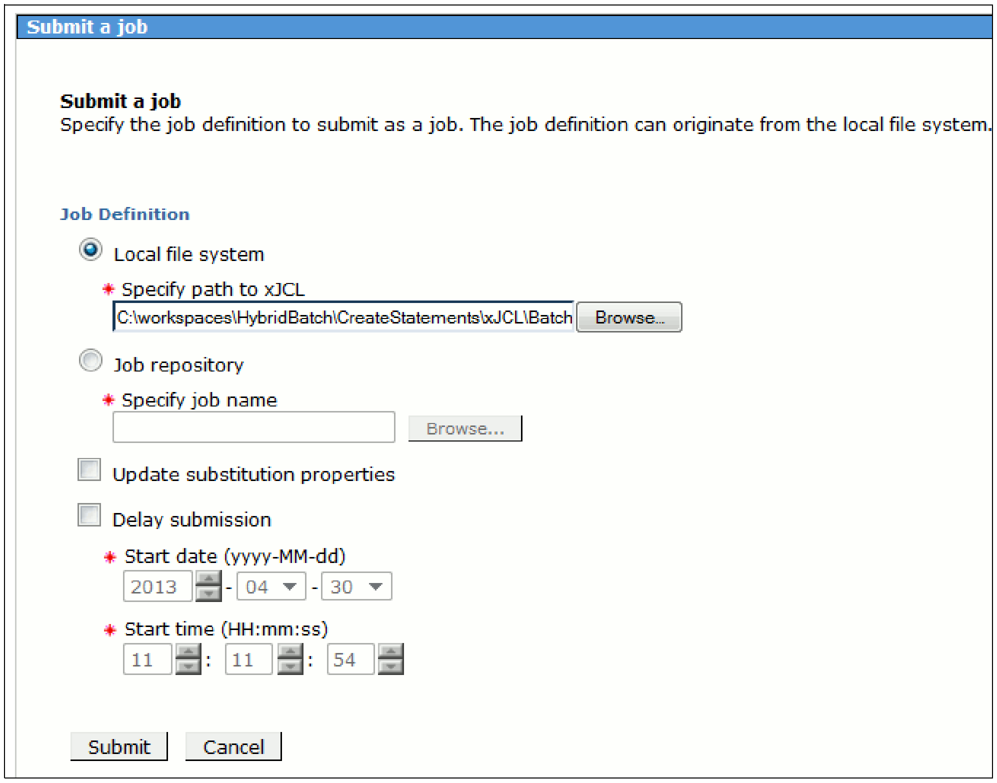
First, export the application as a Java archive (JAR) file that contains the compiled Java classes. Then, follow these steps to install the application in CICS:

1. Copy the JAR file from your workstation onto the mainframe. We used FTP to transfer the file in binary mode. Store the file in a directory that CICS has permission to access. Ensure that CICS has permission to read the JAR file, too.

2. Modify the JVM profile for the JVM server where the batch container is running. Add the JAR file to the `CLASSPATH_SUFFIX`. If you use external interfaces from other JAR files in your batch project, and then these JAR files should also be added to the `CLASSPATH_SUFFIX`.
3. The CICS `JVMSERVER` resource will need to be disabled and re-enabled to pick up the changes to the class path.
4. Modify a configuration file named `batchcontainer-config.xml` to notify the scheduler that CICS can run the batch job.
5. After re-enabling the `JVMSERVER`, run the `CBCH` transaction if it is not already running.

11.3.4 Submit the xJCL to run the batch job

Before you submit the xJCL, make sure that the default application name is the same as the application name that you specified in the *batchcontainer-config.xml* file. Then you can submit this xJCL from any supported enterprise scheduler. For example, if you use the Job Management Console, log on to the web interface, specify the path to the xJCL on your local workstation, and then submit the xJCL to the job scheduler. Figure 11-4 shows how to submit a job from the Job Management Console.


Submit a job

Submit a job
Specify the job definition to submit as a job. The job definition can originate from the local file system.

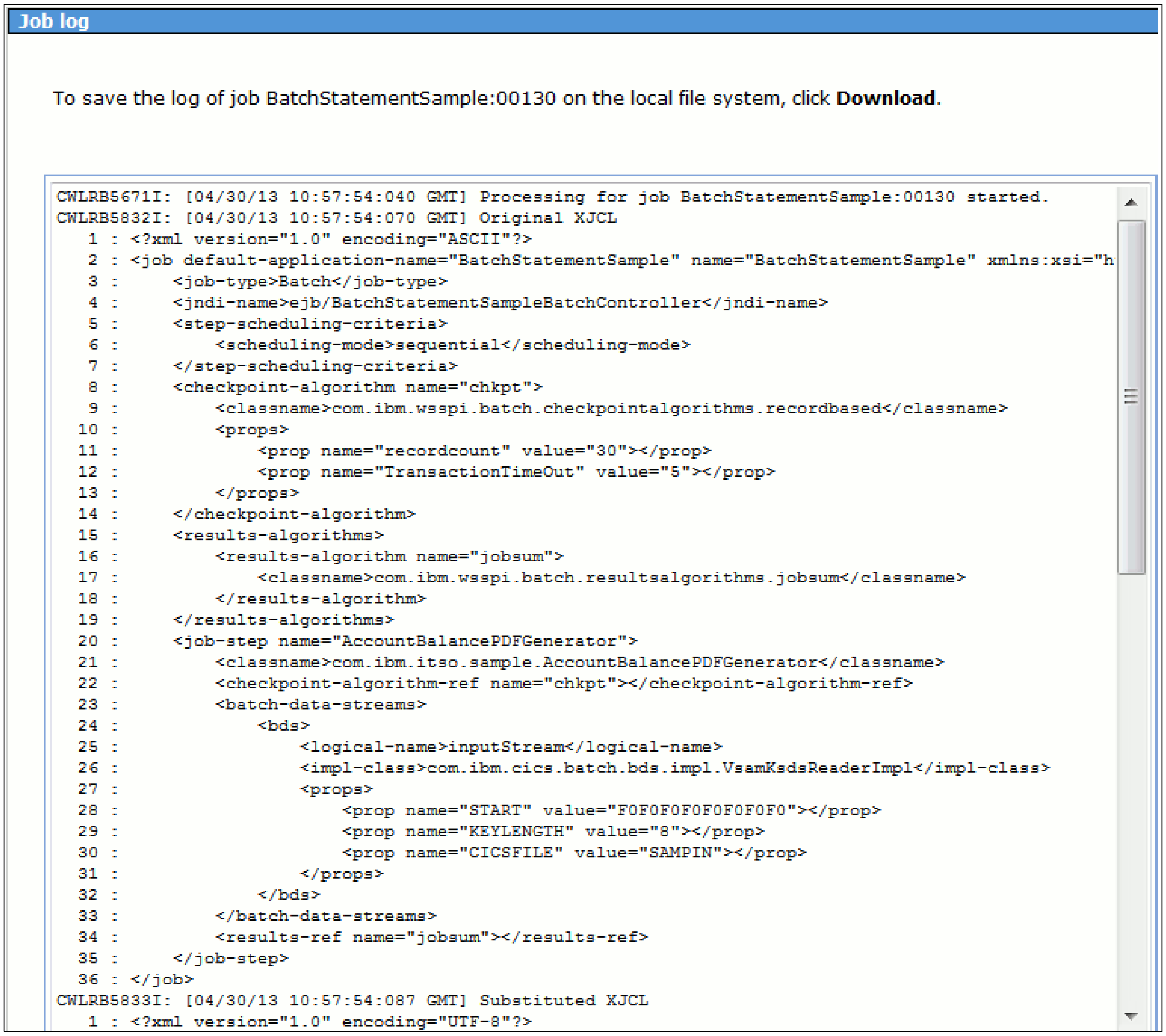
Job Definition

(●) Local file system
* Specify path to xJCL: C:\workspaces\HybridBatch\CreateStatements\xJCL\Batch
( ) Job repository
* Specify job name:
[ ] Update substitution properties
[ ] Delay submission
* Start date (yyyy-MM-dd): 2013 - 04 - 30
* Start time (HH:mm:ss): 11 : 11 : 54

Figure 11-4 Submitting a job from the Job Management Console

A message confirms that the job has been submitted successfully and provides the job ID. When a job is successfully submitted, the job scheduler has identified at least one batch container that claims to be able to run the job. The scheduler chooses a batch container and dispatches the job to it.

You can examine the log for the job to see which batch container the scheduler has chosen to dispatch the job to. Figure 11-5 shows a job log from the Job Management Console.


Job log

To save the log of job BatchStatementSample:00130 on the local file system, click **Download**.

```
CWLRB5671I: [04/30/13 10:57:54:040 GMT] Processing for job BatchStatementSample:00130 started.
CWLRB5832I: [04/30/13 10:57:54:070 GMT] Original XJCL
   1 : <?xml version="1.0" encoding="ASCII"?>
   2 : <job default-application-name="BatchStatementSample" name="BatchStatementSample" xmlns:xsi="h
   3 :     <job-type>Batch</job-type>
   4 :     <jndi-name>ejb/BatchStatementSampleBatchController</jndi-name>
   5 :     <step-scheduling-criteria>
   6 :         <scheduling-mode>sequential</scheduling-mode>
   7 :     </step-scheduling-criteria>
   8 :     <checkpoint-algorithm name="chkpt">
   9 :         <classname>com.ibm.wsspi.batch.checkpointalgorithms.recordbased</classname>
  10 :         <props>
  11 :             <prop name="recordcount" value="30"></prop>
  12 :             <prop name="TransactionTimeOut" value="5"></prop>
  13 :         </props>
  14 :     </checkpoint-algorithm>
  15 :     <results-algorithms>
  16 :         <results-algorithm name="jobsum">
  17 :             <classname>com.ibm.wsspi.batch.resultsalgorithms.jobsum</classname>
  18 :         </results-algorithm>
  19 :     </results-algorithms>
  20 :     <job-step name="AccountBalancePDFGenerator">
  21 :         <classname>com.ibm.itso.sample.AccountBalancePDFGenerator</classname>
  22 :         <checkpoint-algorithm-ref name="chkpt"></checkpoint-algorithm-ref>
  23 :         <batch-data-streams>
  24 :             <bds>
  25 :                 <logical-name>inputStream</logical-name>
  26 :                 <impl-class>com.ibm.cics.batch.bds.impl.VsamKsdsReaderImpl</impl-class>
  27 :                 <props>
  28 :                     <prop name="START" value="F0F0F0F0F0F0F0F0"></prop>
  29 :                     <prop name="KEYLENGTH" value="8"></prop>
  30 :                     <prop name="CICSFILE" value="SAMPIN"></prop>
  31 :                 </props>
  32 :             </bds>
  33 :         </batch-data-streams>
  34 :         <results-ref name="jobsum"></results-ref>
  35 :     </job-step>
  36 : </job>
CWLRB5833I: [04/30/13 10:57:54:087 GMT] Substituted XJCL
   1 : <?xml version="1.0" encoding="UTF-8"?>
```

Figure 11-5 The job log view in the Job Management console

These steps show how to develop a batch application, to deploy it and to schedule it, see the detailed steps in the IBM Redbooks publication titled *New Ways of Running Batch Applications on z/OS: Volume 1 CICS Transaction Server*, SG24-7991:

http://www.redbooks.ibm.com/abstracts/sg247991.html?Open

Related publications

The publications listed in this section are considered particularly suitable for more detailed information about the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information. Some publications referenced in this list might be available in softcopy only.

- *Configuring and Deploying Open Source with WebSphere Application Server Liberty Profile, SG24-8194*
- *Flexible Decision Automation for Your zEnterprise with Business Rules and Events*, SG24-8014
- *IBM CICS and the JVM server: Developing and Deploying Java Applications*, SG24-8038
- *Implementing IBM CICS JSON Web Services for Mobile Applications*, SG24-8161
- *New Ways of Running Batch Applications on z/OS: Volume 1 CICS Transaction Server*, SG24-7991
- *Securing Your Mobile Business with IBM Worklight*, SG24-8179
- *WebSphere Application Server Liberty Profile Guide for Developers*, SG24-8076
- *WebSphere Application Server V8.5 Administration and Configuration Guide for Liberty Profile*, SG24-8170

You can search for, view, download, or order these publications and other Redbooks, Redpapers, Web Docs, drafts, and additional materials on the Redbooks web page:

ibm.com/redbooks

Online resources

These web pages are also relevant for further information:

- CICS Transaction Server for z/OS Value Unit Edition

  http://www.ibm.com/software/products/en/cics-ts-vue
- CICS Transaction Server for z/OS Feature Pack for Dynamic Scripting V2.0

  http://ibm.co/1tAFmTt
- CICS Transaction Server: JEE application role security, IBM Knowledge Center

  http://ibm.co/1pOCfed
- CICS Transaction Server: Liberty features, IBM Knowledge Center

  http://ibm.co/1yD5Ed6
- CICS Transaction Server: The Liberty server angel process, IBM Knowledge Center

  http://ibm.co/12nSggY

- IBM Offering information page (announcement letters and sales manuals)

  `http://www.ibm.com/common/ssi/index.wss?request_locale=en`
- IBM System z Software Pricing

  `http://www.ibm.com/systems/z/resources/swprice`
- IBM System z Software Pricing: System z New Application License Charges (zNALC)

  `http://www.ibm.com/systems/z/resources/swprice/mlc/znalc.html`
- WebSphere Application Server for z/OS: Developing servlets, IBM Knowledge Center

  `http://ibm.co/1tJ4C94`
- WebSphere Application Server Liberty Core: Authentication, IBM Knowledge Center

  `http://ibm.co/1tAGVAI`
- WebSphere Application Server Liberty Core: Configuration elements in the server.xml file, IBM Knowledge Center

  `http://ibm.co/11PC7AO`
- IBM WebSphere Application Server Migration Toolkit, IBM developerWorks

  `http://www.ibm.com/developerworks/websphere/downloads/migtoolkit/`
- WebSphere Application Server Network Deployment: Bean validation, IBM Knowledge Center

  `http://ibm.co/1HW5PnX`
- WebSphere Application Server Network Deployment: Developing JSP files, IBM Knowledge Center

  `http://ibm.co/15NYOal`
- WebSphere Application Server Network Deployment: Developing web services - RESTful services, IBM Knowledge Center

  `http://ibm.co/1AcAZ5v`
- WebSphere Application Server Network Deployment: Java Architecture for XML Binding (JAXB), IBM Knowledge Center

  `http://ibm.co/1yJ4xbM`
- WebSphere Application Server Network Deployment: JavaScript Object Notation (JSON4J), IBM Knowledge Center

  `http://ibm.co/1ycEawg`
- WebSphere Application Server Network Deployment: JavaServer Faces, IBM Knowledge Center

  `http://ibm.co/1HW3QjC`
- WebSphere Application Server Network Deployment: JAX-WS, IBM Knowledge Center

  `http://ibm.co/1BakfiJ`

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM Redbooks

A Software Architect's Guide to Java Workloads in IBM CICS Transaction Server

(0.2"spine)
0.17"<->0.473"
90<->249 pages

A Software Architect's Guide to New Java Workloads in IBM CICS Transaction Server

Review the architectural and technical benefits of this approach

Get details about new licensing models at attractive prices

Learn about web and mobile business rules and batch workload processes

This IBM Redbooks publication introduces the IBM System z New Application License Charges (zNALC) pricing structure and provides examples of zNALC workload scenarios. It describes the products that can be run on a zNALC logical partition (LPAR), reasons to consider such an implementation, and covers the following topics:

- Using the IBM WebSphere Application Server Liberty Profile to host applications within an IBM CICS environment and how it interacts with CICS applications and resources
- Security technologies available to applications that are hosted within a WebSphere Application Server Liberty Profile in CICS
- How to implement modern presentation in CICS with a CICS Liberty Java virtual machine (JVM) server
- How to share scenarios to develop Liberty JVM applications to gain benefits from IBM CICS Transaction Server for z/OS Value Unit Edition
- Considerations when using mobile devices to interact with CICS applications and explains specific CICS technologies for connecting mobile devices by using the z/OS Value Unit Edition
- How IBM Operational Decision Manager for z/OS runs in the transaction server to provide decision management services for CICS COBOL and PL/I applications
- Installing the CICS Transaction Server for z/OS Feature Pack for Modern Batch to enable the IBM WebSphere batch environment to schedule and manage batch applications in CICS

This book also covers plain old Java objects (POJOs). The CICS JVM server is a full-fledged JVM that includes support for Open Service Gateway initiative (OSGi) bundles. POJO applications can also qualify for using the Value Unit Edition.

SG24-8225-00 ISBN 0738440256