

Liberty in IBM CICS

Deploying and Managing Java EE Applications

Phil Wakelin

Carlos Donatucci

Jonathan Lawrence

Mitch Johnson

Michael Jones

Tito Paiva



z Systems



International Technical Support Organization

Liberty in IBM CICS: Deploying and Managing Java EE Applications

January 2018

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

First Edition (January 2018)

This edition applies to Version 5, Release 4 of IBM CICS Transaction Server.

© Copyright International Business Machines Corporation 2018. All rights reserved.

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

Contents

Notices	vii
Trademarks	viii
Preface	ix
Authors	ix
Now you can become a published author, too!	x
Comments welcome	xi
Stay connected to IBM Redbooks	xi
Chapter 1. Installation and configuration	1
1.1 Getting your CICS region ready	2
1.2 zFS file system configuration	2
1.2.1 zFS configuration files	2
1.2.2 Autoconfiguration	3
1.2.3 zFS structure	3
1.2.4 zFS output files	5
1.2.5 zFS file permissions	5
1.2.6 zFS file encodings and editing tools	7
1.3 Setting up a Liberty JVM Server	11
1.3.1 JVM profile	11
1.3.2 Tailoring the JVM profile	11
1.3.3 Liberty specific options	14
1.3.4 IBM Language Environment runtime options	15
1.3.5 Creating a JVMSERVER resource	16
1.4 Tailoring server.xml	18
1.4.1 Adding Liberty features	18
1.4.2 Include files	19
1.4.3 Configuring the HTTP and HTTPS endpoints	19
1.4.4 CICS bundle deployed applications	21
1.4.5 Default web application	21
1.4.6 Liberty transaction log files	23
1.4.7 Sample server.xml file	24
1.4.8 Welcome page	25
Chapter 2. Deploying a web application	27
2.1 Building the restapp sample	28
2.1.1 Obtaining the sample code	28
2.1.2 Creating the Eclipse projects	29
2.2 Deploying a web application to Liberty	32
2.2.1 Deployment by using Liberty dropins	33
2.2.2 Deployment as an application element in server.xml	37
2.2.3 Deployment in a CICS bundle	40
2.2.4 Comparison of the deployment options	52
2.3 Advanced deployment options	53
2.3.1 Shared libraries	53
2.3.2 Global libraries	54
2.3.3 Deploying a prebuilt Java archive in a CICS bundle	54
2.3.4 Pausing and resuming a server	58

Chapter 3. Link to Liberty	61
3.1 Overview	62
3.1.1 Prerequisites	63
3.1.2 How it works	63
3.2 Link to Liberty sample application	65
3.2.1 Building the sample application	65
3.2.2 Sample application	72
3.2.3 Deploying the sample	74
3.2.4 Running the sample	78
3.2.5 Manual program definition	80
3.2.6 Updating a Link to Liberty program	81
3.3 Qualities of service	82
3.3.1 Transactions	82
3.3.2 Exception and abend processing	83
3.3.3 Security	86
3.3.4 Summary	89
 Chapter 4. Connecting to Db2 by using JDBC	 91
4.1 JDBC overview	92
4.1.1 JDBC drivers	92
4.1.2 Data sources	93
4.1.3 Static and dynamic SQL	94
4.2 Installing the JDBC Employee application	94
4.2.1 Liberty features	95
4.2.2 Data source definition	95
4.2.3 CICS resources	95
4.3 Using JDBC type 2 connectivity	97
4.3.1 Configuring CICS resources	98
4.3.2 Configuring server.xml	100
4.3.3 Binding the plan	102
4.3.4 Running the application	103
4.4 Using JDBC type 4 connectivity	104
4.4.1 Configuring CICS resources	104
4.4.2 Configuring server.xml	105
4.4.3 Running the application	107
4.4.4 Container managed security	107
4.5 Transactional support with JDBC	108
4.6 Tracing JDBC	110
 Chapter 5. Connecting to IBM MQ by using JMS	 113
5.1 Introduction to JMS	114
5.1.1 Java Message Service	114
5.1.2 Message Driven Beans	114
5.1.3 Java Naming and Directory Interface	114
5.1.4 Connection pooling	116
5.2 JMS sample application	116
5.2.1 Modifying the JMS sample application	117
5.2.2 Deploying the JMS sample application	119
5.2.3 Configuring Liberty for the JMS sample application	119
5.2.4 Describing the JMS updates to the JVM server profile	121
5.3 Required CICS resources	122
5.3.1 BUNDLE resources	122
5.3.2 URIMAP resource	123
5.3.3 Transaction resources	124

5.4 Required IBM MQ resources	125
5.4.1 Configuring IBM MQ Explorer	125
5.4.2 Defining the queues	125
5.5 Testing the sample applications	126
5.5.1 Testing the MQJMSDemo application.	126
5.5.2 Testing the MySimpleMDB application	128
5.5.3 Use of the Execution Diagnostic Facility	129
5.6 Security	129
5.6.1 RACF resources	129
5.6.2 JMS security scenarios	131
5.6.3 Summary.	135
5.7 Transport Layer Security	136
5.7.1 RACF resources	136
5.7.2 TLS debugging hints and tips	139
Chapter 6. Configuring Transport Layer Security support	141
6.1 JSSE and JCE	142
6.1.1 Updating the JCE policy files	142
6.2 TLS server authentication by using a Java keystore.	143
6.3 TLS server authentication by using a RACF key ring	149
6.4 TLS client authentication.	152
6.5 Hints and tips when using TLS	156
6.5.1 Tracing TLS	156
6.5.2 Enforcing TLS for web applications.	157
6.5.3 HTTP persistent connections	158
6.5.4 TLS session timeout	160
6.5.5 Controlling the TLS version.	160
6.5.6 Controlling the cipher suite	161
6.5.7 Restricting weak algorithms	162
6.6 Using cryptographic hardware with JSSE	164
6.6.1 Cryptographic hardware	164
6.6.2 Cryptographic software	165
6.6.3 Configuring TLS to use the cryptographic coprocessors	167
6.6.4 Monitoring cryptographic hardware	168
Chapter 7. Securing web applications	175
7.1 Overview	176
7.2 z/OS security configuration for Liberty JVM servers	177
7.2.1 Starting the angel process	177
7.2.2 Setting up access to the angel process	177
7.2.3 Profile prefix and required SAF profiles	179
7.2.4 SAF profile summary.	180
7.3 Configuring a Liberty security registry	180
7.3.1 Configuring a basic user registry.	181
7.3.2 Configuring a SAF registry	182
7.3.3 Configuring an LDAP registry	184
7.4 Authentication scenarios	186
7.4.1 Basic authentication with a SAF registry.	187
7.4.2 Basic authentication by using LDAP credentials.	190
7.4.3 Form-based login	193
7.4.4 Certificate-based client authentication	197
7.5 Authorization scenarios.	201
7.5.1 URL-specific authorization by using EJBROLES.	201

7.5.2	Programmatic role authorization by using EJBROLEs	206
7.5.3	CICS transaction security with URIMAPs	211
7.6	Configuring SSO by using Lightweight Third-Party Authentication	214
7.6.1	Configuring LTPA	215
7.6.2	Disabling SSO in Liberty	217
7.6.3	Requiring TLS when using SSO	217
7.7	JSON client code with cookie printer	217
Chapter 8.	Logging and monitoring	221
8.1	Message and log files	222
8.1.1	CICS logs	222
8.1.2	Java logs	223
8.1.3	Liberty server logs	224
8.1.4	JVM server trace output	228
8.2	Monitoring tools	232
8.2.1	CICS statistics records	232
8.2.2	CICS performance records	234
8.2.3	CICS Performance Analyzer	236
8.2.4	CICS Explorer	238
8.2.5	IBM Health Center	239
8.2.6	IBM Application Metrics for Java	244
8.2.7	Runaway tasks	247
8.2.8	CICS policies	248
Chapter 9.	Port sharing and cloning regions	253
9.1	Sharing ports	254
9.1.1	Using WLMHEALTH	256
9.2	Cloning regions	258
9.2.1	Sharing application definitions	258
9.2.2	Sharing SSL configuration	260
9.2.3	Sharing feature configuration	261
9.2.4	Sharing LTPA keys	262

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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

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

IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

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

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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


COPYRIGHT LICENSE:

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

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

CICS®	IBM z13®	Redpapers™
CICS Explorer®	IMS™	Redbooks (logo)  ®
CICSplex®	Language Environment®	RMF™
Db2®	MVS™	WebSphere®
DB2®	RACF®	z/OS®
developerWorks®	Redbooks®	z13®
IBM®	Redpaper™	

The following terms are trademarks of other companies:

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

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

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

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

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

Preface

This IBM® Redbooks® publication is intended for IBM CICS® system programmers and IBM Z architects. It describes how to deploy and manage Java EE 7 web-based applications in an IBM CICS Liberty JVM server and access data on IBM Db2® for IBM z/OS® and IBM MQ for z/OS sub systems.

In this book, we describe the key steps to create and install a Liberty JVM server within a CICS region. We then describe how to best use the different deployment techniques for Java EE applications and the specific considerations when deploying applications that use JDBC, JMS, and the new CICS link to Liberty API.

Finally, we describe how to secure web applications in CICS Liberty, including transport-level security and request authentication and authorization by using IBM RACF® and LDAP registries. Information is also provided about how to build a high availability infrastructure and how to use the logging and monitoring functions that are available in the CICS Liberty environment.

This book is based on IBM CICS Transaction Server (CICS TS) V5.4 that uses the embedded IBM WebSphere® Application Server Liberty technology. It is also applicable to CICS TS V5.3 with the fixes for the continuous delivery APAR PI77502 applied. Sample applications are used throughout this publication and are freely available for download from the IBM CICSDev GitHub organization along with detailed deployment instructions.

Authors

This book was produced by a team of specialists from around the world.

Phil Wakelin works for CICS development at IBM UK in Hursley, and is a member of the CICS strategy and design team. He has worked with many CICS technologies for the last 25 years, and is currently responsible for Java Adoption with new CICS customers. He is the author of many white papers, SupportPacs, and IBM Redbooks publications in the areas of CICS integration and Java support.

Carlos Donatucci is a certified IT specialist expert working at IBM Buenos Aires Global Delivery Center as CICS Subject Matter Expert specialist for several accounts in the US and Canada. He is a member of the Americas Certification Board and the Technical Counsel of Argentina. He has more than 27 years working as a Mainframe specialist in several areas, such as software management, automation, scheduling, and online and batch optimization.

Jonathan Lawrence is a Software Engineer in the CICS Level 3 Service team at IBM Hursley UK, specializing in Java, Liberty, and related technologies. He previously held roles as an IBM Software Services CICS integration specialist and in the Java Technology Center. He was also an author of another IBM Redbooks publication.

Mitch Johnson is currently working as a Subject Matter Expert for IBM MQ, z/OS Connect EE, and ODM in the Washington System Center in the United States. Before this role, Mitch was a consultant in IBM Software Services for WebSphere where he was a Subject Matter Expert for WebSphere Application Server on z/OS. He has worked on previous IBM Redbooks publications and IBM Redpapers™ publications that primarily focused on connectivity to z/OS resources from WebSphere Application Server and other resources. His areas of expertise include CICS; IBM DB2®; IBM IMS™; IBM MQ; z/OS; Java; Java Platform, Enterprise Edition Connectors; and security.

Michael Jones is a Software Engineer working at IBM Hursley UK. He has over 10 years of experience in Java. He also works as the lead tester for Java in CICS and as a Java SME on IBM CICS TS modernization projects.

Tito Paiva has worked as a developer, DBA, and application development support since the 1980s. He worked for 5 years as a WebSphere for z/OS pre-sales specialist at IBM SWG in Brazil. He now is working as a Subject Matter Expert for middleware at IBM GTS in Poland. His areas of expertise include WebSphere, Java, DB2, CICS, IMS, IBM MQ, and security.

This IBM Redbooks publication was managed and edited by Martin Keen.

Special thanks to Robert Haimowitz for his outstanding support.

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

- ▶ Ivan Hargreaves
- ▶ Nigel Williams
- ▶ Eric Phan
- ▶ Jason Katonica
- ▶ Eysha Powers
- ▶ Ian Burnett
- ▶ David Roberts
- ▶ Christopher Meyer
- ▶ Matthew Wilson
- ▶ Matt Leming
- ▶ Lyn Elkins
- ▶ Alexander Brown
- ▶ Mark Cocker
- ▶ Andy Wharmby

Now you can become a published author, too!

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- Send your comments in an email to:

redbooks@us.ibm.com

- Mail your comments to:

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

Stay connected to IBM Redbooks

- Find us on Facebook:

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

- Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- Look for us on LinkedIn:

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

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

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

- Stay current on recent Redbooks publications with RSS Feeds:

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



Installation and configuration

In this chapter, we describe how we configured our Liberty JVM server in CICS to run our web applications during this IBM Redbooks publication project. This configuration process includes modifying our CICS region and setting up UNIX System Services resources.

This chapter includes the following topics:

- ▶ 1.1, “Getting your CICS region ready” on page 2
- ▶ 1.2, “zFS file system configuration” on page 2
- ▶ 1.3, “Setting up a Liberty JVM Server” on page 11
- ▶ 1.4, “Tailoring server.xml” on page 18

1.1 Getting your CICS region ready

The first step to run Java in CICS TS is to ensure that the UNIX System Services and Java environment are enabled in CICS. Complete the following steps:

1. Add the SDFJAUTH library to the CICS region's STEPLIB:

```
//STEPLIB DD DISP=SHR,DSN=CICSTS54.CICS.SDFHAUTH
//          DD DISP=SHR,DSN=CICSTS54.CICS.SDFJAUTH
```

Tip: You also must ensure that SDFJAUTH and SEYUAUTH libraries are APF-authorized in the same manner as used for the SDFHAUTH library.

2. If you are planning a CMCI stand-alone connection with IBM CICS Explorer®, you should also include the SEYUAUTH library in the STEPLIB concatenation:

```
//          DD DISP=SHR,DSN=CICSTS54.CICS.SEYUAUTH
```

3. Set the USSHOME SIT parameter to the location where the CICS UNIX System Services libraries are installed. In our example, we use the following location:

```
USSHOME=/usr/lpp/cicsts/cicsts54
```

4. Determine where your Java runtime is installed. For the purposes of our environment, our setting is </usr/lpp/java/J8.0_64_SR4>.

1.2 zFS file system configuration

Liberty bases its configuration around zFS files. It is important to understand how these files are structured and used.

1.2.1 zFS configuration files

The following zFS configuration files are required to run a Liberty JVM server in CICS:

1. **JVM profile:** This configuration file is for the JVM server. It must be Extended Binary Coded Decimal Interchange Code (EBCDIC) that is encoded and includes the environment variables and JVM settings that are necessary to support the JVM server.

The JVM profile is located by using the JVMPROFILEDIR system initialization parameter for your CICS region. CICS provides a set of fully commented examples in the USSHOME/JVMProfiles zFS directory. In our example, we show you how to configure your Liberty JVM server by using the sample JVM profile and editing it for a first startup.

We set the following JVMPROFILEDIR location:

```
/var/cicsts/&applid;/JVMProfiles
```

This location is resolved to the following zFS directory:

```
/var/cicsts/SC8CICS7/JVMProfiles
```

2. **server.xml:** This configuration file is for the Liberty server. It is encoded in ASCII and is an XML-based configuration file. It is normally created on the first startup of the Liberty JVM server by using autoconfigure support; then, it can be edited at any time to introduce modifications.

We set the location of our `server.xml` by using the `WLP_USER_DIR` option in the JVM server profile:

```
WLP_USER_DIR=/var/cicsts/&APPLID;/wlp
```

This setting caused our `server.xml` to be created in the following location:

```
/var/cicsts/SC8CICS7/wlp/servers/itsowlp1/server.xml
```

1.2.2 Autoconfiguration

The `server.xml` configuration files define the Liberty configuration that you want to use for your Liberty JVM server. If you create a Liberty JVM server for the first time with the `autoconfigure` option enabled in your JVM profile, this file is created for you with the rest of the Liberty server zFS structure. We suggest that you start your Liberty JVM servers with this option enabled for at least the first startup of the Liberty JVM server.

To enable autoconfiguration, set the following JVM system property to `true` in your JVM profile:

```
-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true
```

1.2.3 zFS structure

We defined the zFS structure as follows in our IBM Redbooks project. The root directory for our zFS is defined as `/var/cicsts/<applid>/`, under which the following directories were defined:

- ▶ `ITS0JVM1`: JVM server output files and application deployment directories.
- ▶ `JVMProfiles`: JVM server configuration profiles.
- ▶ `wlp`: Liberty server configuration and output files.

This configuration provides all of the main subdirectories under a single parent directory so that they can be easily traversed.

Our file system structure is shown in Figure 1-1, which shows the directories under /var/cicsts/SC8CICS7/ root directory.

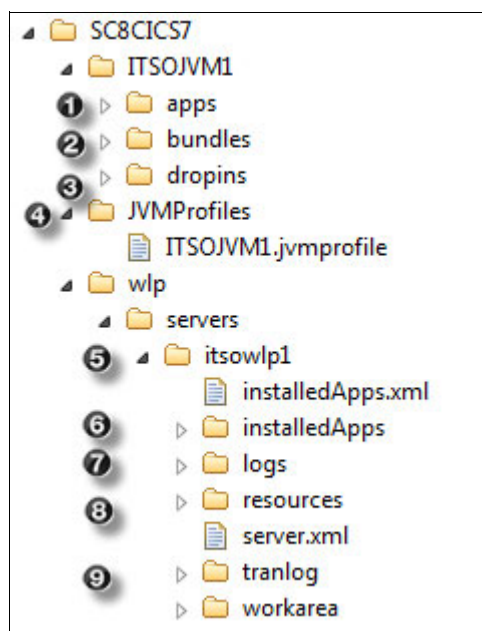


Figure 1-1 File system structure of /var/cicsts

The following directories are included in our zFS configuration:

- ▶ ITS0JVM1/apps: This directory is created for Liberty applications that are deployed by using the <application> element.
- ▶ ITS0JVM1/bundles: This directory is where the output for exported CICS bundle projects is stored.
- ▶ ITS0JVM1/dropins: This directory is used for applications that are deployed by using the drop-ins mechanism.

Tip: For more information about the location that is used for deploying applications, see Chapter 2, “Deploying a web application” on page 27.

- ▶ JVMProfiles: This directory includes our JVM profile ITS0JVM1.jvmprofile for our JVMSERVER ITS0JVM1.
- ▶ wlp/servers/itsowlp1: This directory is the Liberty \${server.config.dir} and \${server.config.dir} for Liberty configuration and output files.
- ▶ wlp/servers/itsowlp1/installedApps: This directory is used as a temporary installation area for applications that are installed by CICS bundles. When a CICS bundle is enabled, CICS installs any Liberty web applications into the \${server.output.dir}/installedApps directory.
- ▶ wlp/servers/itsowlp1/logs: Liberty output logs are written to this directory, including FFDC, messages.log, and the optional HTTP access log.
- ▶ wlp/servers/itsowlp1/resources: Resources that are related to security administration are stored in this directory.
- ▶ wlp/servers/itsowlp1/tranlog: The Liberty transaction manager stores its recoverable log files in the zFS filing system in this directory. An entry in the server.xml file modifies this location, if required.

Note: The zFS file system settings that are used in our scenario were chosen to shorten the directory paths, which makes it easier to browse to the related zFS artifacts for each CICS region.

1.2.4 zFS output files

Several output files are used by the JVM server, such as the stdout, stderr, JVM server trace, Liberty first-failure data capture (FFDC), and message logs. These files are stored in region-specific subdirectories that are based on the WORK_DIR and WLP_USER_DIR settings in the JVM profile. Because we used WORK_DIR=/var/cicsts and WLP_USER_DIR=/var/cicsts/&APPLID;/wlp in our example, the zFS output files are in the /var/cicsts/SC8CICS7/wlp directory, as shown in Figure 1-2.

Menu Utilities View Options Help												
z/OS UNIX Directory List											Row 1 to 12 of 12	
Command ==> █											Scroll ==> CSR	
Pathname . : /cicsts/SC8CICS7/wlp/servers/itsowlp1												
EUID . . . : 191												
Command	Filename	Message	Type	Permission	Audit	Ext	Fmat	Owner	Group	Links	Size	Modified
_____	.		Dir	rw-rw-r-x	fff--		----	CICSREGN	CICSADMN	9	8192	2017/10/13 18:21:02
_____	..		Dir	rw-rw-r-x	fff--		----	CICSREGN	CICSADMN	3	8192	2017/10/06 14:47:11
_____	apps		Dir	rw-r--r-x	fff--		----	MICHAEL	CICSADMN	2	8192	2017/10/13 15:22:06
_____	dropins		Dir	rw-rw-r-x	fff--		----	CICSREGN	CICSADMN	2	8192	2017/10/13 13:46:08
_____	installedApps		Dir	rw-rw-r-x	fff--		----	CICSREGN	CICSADMN	2	8192	2017/10/13 18:21:06
_____	installedApps.x		File	rw-rw-r-x	fff--	--s-	----	CICSREGN	CICSADMN	1	1898	2017/10/13 18:21:06
_____	jms.xml		File	rw-x-----	fff--	--s-	----	CICSREGN	CICSADMN	1	1484	2017/10/10 14:39:10
_____	logs		Dir	rw-rw-r-x	fff--		----	CICSREGN	CICSADMN	4	8192	2017/10/13 18:21:03
_____	resources		Dir	rw-rw-r-x	fff--		----	CICSREGN	CICSADMN	3	8192	2017/10/06 14:47:17
_____	server.xml		File	rw-rw-r-x	fff--	--s-	----	CICSREGN	CICSADMN	1	2227	2017/10/13 18:33:12
_____	tranlog		Dir	rw-rw-r-x	fff--		----	CICSREGN	CICSADMN	4	8192	2017/10/06 14:47:18
_____	workarea		Dir	rw-rw-r-x	fff--		----	CICSREGN	CICSADMN	5	8192	2017/10/13 18:21:03
***** Bottom of data *****												

Figure 1-2 zFS output files

1.2.5 zFS file permissions

Access to zFS files in a JVM server is granted by using the CICS region user ID and the UNIX System Services file permission bits. You must ensure that the CICS region user ID includes a UNIX System Services segment, and that the CICS region user ID or the group it belongs to include the following permissions:

- Read, write, and execute access to the JVM server working directory for creating log files and the Liberty server configuration files.
- Read and execute access to the directory tree that contains the JVM profiles.
- Read access to the JVM profile.

Note: In UNIX environments, read and execute access to all directories in the tree is required to enter a directory and read a file in the specific directory.

In our environment, we used the RACF groups and user IDs that are listed in Table 1-1.

Table 1-1 User IDs and group IDs

User IDs	Group ID	Description
CICSREGN	CICSPROD	CICS region user ID
CARLOS, PHIL	ADMIN	Administrator user IDs
WEBUSER, LDAPUSER	CICSUSRS	User IDs for web application authorization, as described in Chapter 7, “Securing web applications ” on page 175

By using these credentials, we issued the following UNIX System Services commands to grant the CICS administrators group (CICSADMN) and the CICS region user ID (CICSREGN) read/write access to the working directory structure, and all other users read access only:

```
chown CICSREGN /var/cicsts/SC8CICS7
chgrp ADMIN /var/cicsts/SC8CICS7
chmod 775 /var/cicsts/SC8CICS7
chown CICSREGN /var/cicsts/SC8CICS7/ITS0JVM1
chgrp ADMIN /var/cicsts/SC8CICS7/ITS0JVM1
chmod 775 /var/cicsts/SC8CICS7/ITS0JVM1
```

The output of the `ls -l` command for the working directory `/var/cicsts/SC8CICS7/ITS0JVM1` that shows the assigned permissions `rw-rwxr-x` is shown in Figure 1-3.

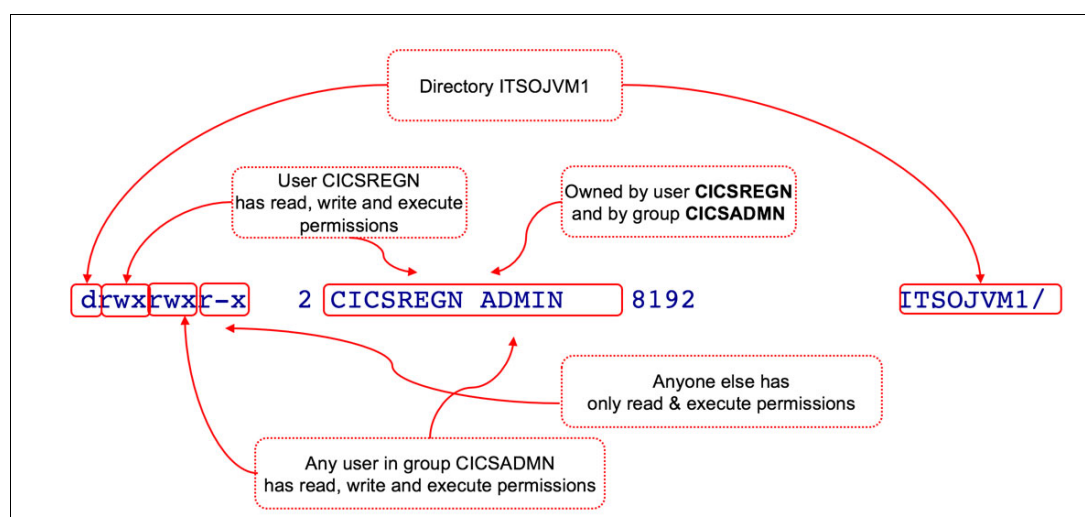


Figure 1-3 UNIX directory permissions

We then granted the CICS region user ID read access and the CICS administrators group read/write access to the JVM profile, as shown in the following example:

```
chown -R CICSREGN /var/cicsts/SC8CICS7/JVMProfiles
chgrp -R ADMIN /var/cicsts/SC8CICS7/JVMProfiles
chmod 570 /var/cicsts/SC8CICS7/JVMProfiles
chmod 460 /var/cicsts/SC8CICS7/JVMProfiles/ITS0JVM1.jvmprofile
```

User file creation mask (UMASK)

The JVM profile option `_DFH_UMASK` sets the UNIX System Services process UMASK that applies when any files are created by a JVM server. This value is a three-digit octal, which determines the permission bits that are *not* set.

The default value of 007 allows the intended read/write/execute permissions of owner and group to be respected, while preventing read/write/execute being granted to other users when a file is created.

In our scenario, we set `_DFH_UMASK=002`, which gives `rw-rw-r--` permissions when files are created and `rxwxrwxr-x` permission for directories. This configuration allows the CICS region and the administrators to view and manage the log files and other users that are not in the ADMIN group to view the log files.

The z/OS UNIX directory list that displays the permission bits for the `/var/cicsts/SC8CICS7/wlp/servers/itsowlp1/logs` directory and its files (when this value is used) is shown in Figure 1-4.

z/OS UNIX Directory List									
Command ==> █									
Pathname . : /cicsts/SC8CICS7/wlp/servers/itsowlp1/logs									
EUID . . . : 191									
Command	Filename	Message	Type	Permission	Audit	Ext	Fmat	Owner	Group
_____	.		Dir	rxwxrwxr-x	fff--		----	CICSREGN	CICSADMN
_____	..		Dir	rxwxrwxr-x	fff--		----	CICSREGN	CICSADMN
_____	ffdc		Dir	rxwxrwxr-x	fff--		----	CICSREGN	CICSADMN
_____	http_access.log		File	rw-rw-r--	fff--	--s-	----	CICSREGN	CICSADMN
_____	messages.log		File	rw-rw-r--	fff--	--s-	----	CICSREGN	CICSADMN
_____	messages_17.10.		File	rw-rw-r--	fff--	--s-	----	CICSREGN	CICSADMN
_____	state		Dir	rxwxrwxr-x	fff--		----	CICSREGN	CICSADMN
*****					Bottom of data *****				

Figure 1-4 ISPF option 3.4 logs directory

1.2.6 zFS file encodings and editing tools

The JVM can use a different code page from CICS for character encoding. Although CICS must always use an EBCDIC code page, a Liberty JVM server must use an American Standard Code for Information Interchange (ASCII) encoding, such as `iso8859_1`. This requirement means that the Liberty server configuration and output files are encoded in ASCII, whereas the JVM server profile is encoded in EBCDIC along with any JVM server stdout, stderr, and trace files.

Several different editing tools are available with which you can view zFS files on z/OS. Each of these tools features different mechanisms to use for handling ASCII versus EBCDIC files. Next, we describe the following common tools and options each provides:

- ▶ Enhanced ASCII
- ▶ ISPF
- ▶ File Transfer Protocol (FTP)
- ▶ CICS Explorer
- ▶ UNIX tools

Enhanced ASCII

Enhanced ASCII introduces automatic conversion of z/FS files between ASCII, Unicode, and EBCDIC encodings. It is controlled by using the zFS file tags, which describe an individual file encoding and the conversion behavior controlled that uses the `_BPXK_AUTOCVT=ALL` variable in the user's profile or by setting the `USS BPXPRMxx AUTOCVT` parameter to ALL.

The output of the UNIX System Services `ls -lT` command is shown in Example 1-1. In the example, a JVM profile that is tagged with the EBCDIC encoding IBM-1047 also is shown.

Example 1-1 UNIX System Services list of EBCDIC tagged jvm profile

```
$ [SC80] /cicsts/SC8CICS7: ls -lT JVMProfiles
total 16
t IBM-1047    T=on  -rwxrwxr-x    1 CICSREGN ADMIN 1548 Nov 18 03:59
ITS0JVM1.jvmprofile
```

Tip: Starting with CICS TS V5.3, all CICS zFS files are tagged with the relevant file encoding: ISO8859-1 or IBM-1047.

ISPF option 3.4

The z/OS UNIX directory list function, which is accessed by using ISPF option 3.4, is a popular means for editing files on z/OS. It supports IBM MVS™ datasets and zFS file browsing and editing. By entering the '/' action, you can edit and view files by using EBCDIC, ASCII, and UTF-8 options, as shown in Figure 1-5. This action makes it easy to manage any type of file.

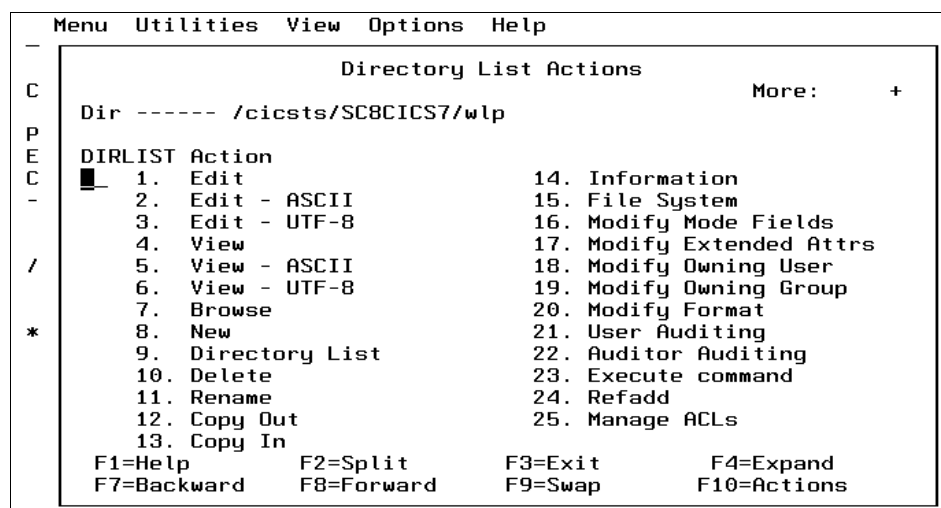


Figure 1-5 ISPF option 3.4 directory list actions

File Transfer Protocol

FTP is a standard protocol for transmitting files between computers over TCP/IP connections. It is used by many third-party tools to transfer files, and can also be started by using the FTP command line utility. The FTP utility can transfer files in *binary* mode, without conversion, or in *ASCII* mode, with conversion between client and server encodings.

If set up correctly, the z/OS FTP utility transfers and converts ASCII and EBCDIC zFS files that are correctly tagged. Any files that are not tagged are converted by using the default settings that are provided by the SBDATACONN in the FTP server in the SYS1.TCPPARMS(FTPDATA) member. In our system, this setting was configured by using the following values, which assume an EBCDIC encoding of IBM-1047 for zFS files and an ASCII encoding of ISO8859-1 for workstation files:

```
SBDATACONN (IBM-1047,ISO8859-1)
```

The FTP transfer of the ASCII encoded server.xml from z/OS to our Windows workstation is shown in Example 1-2.

Example 1-2 FTP transfer of tagged server.xml

```
ftp> get server.xml
200 Port request OK.
125-Tagged ASCII file translated with table built using file system cp=ISO8859-1,
network transfer cp=ISO8859-1
125 Sending data set /var/cicsts/SC8CICS7/wlp/servers/itsowlp1/server.xml
250 Transfer completed successfully.
ftp: 4170 bytes received in 0.09Seconds 47.39Kbytes/sec.
```

CICS Explorer: z/OS perspective

The z/OS UNIX Files view (see Figure 1-6) is provided in the CICS Explorer z/OS perspective. It provides a Windows Explorer-like interface for zFS that is based on FTP connectivity to z/OS. If set up correctly, editing ASCII and EBCDIC files can be managed based on the file encodings that are specified for each file.

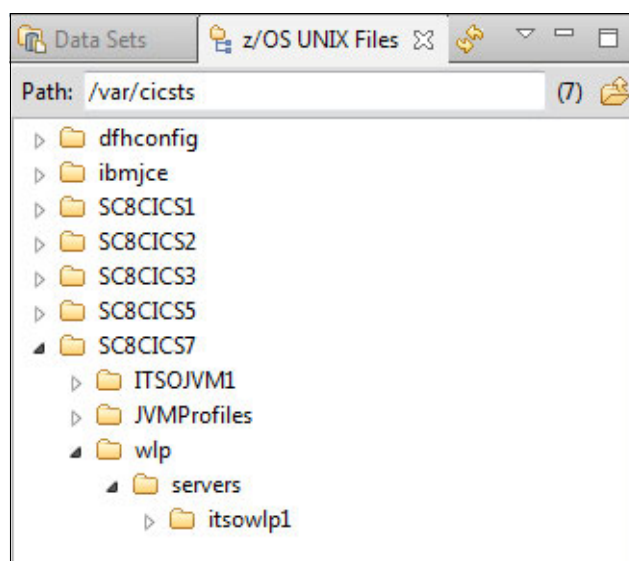


Figure 1-6 z/OS UNIX Files view

To configure this function, the SBADATACONN value in the FTP server or the default EBCDIC encoding for the CICS Explorer FTP connection must be set, as shown in Figure 1-7 on page 10.

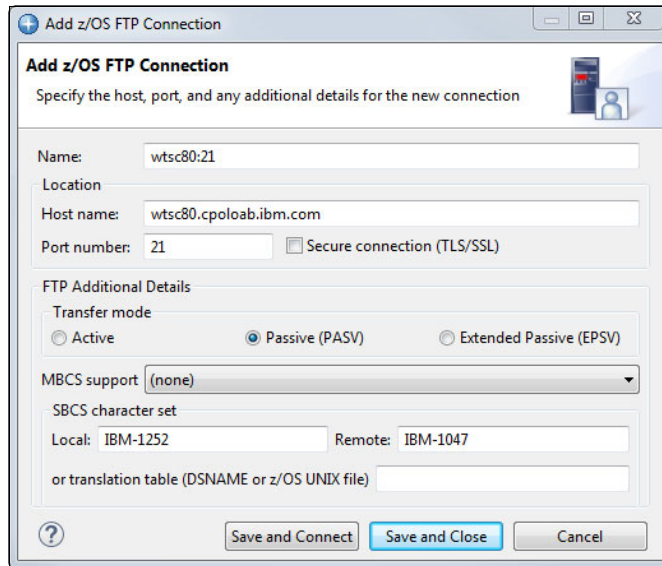


Figure 1-7 z/OS FTP Connection settings

If this Remote character set is not set correctly, each file that contains ASCII content must be individually tagged as being transferred in binary FTP mode to prevent any file conversion during the FTP transfer (see Figure 1-8).

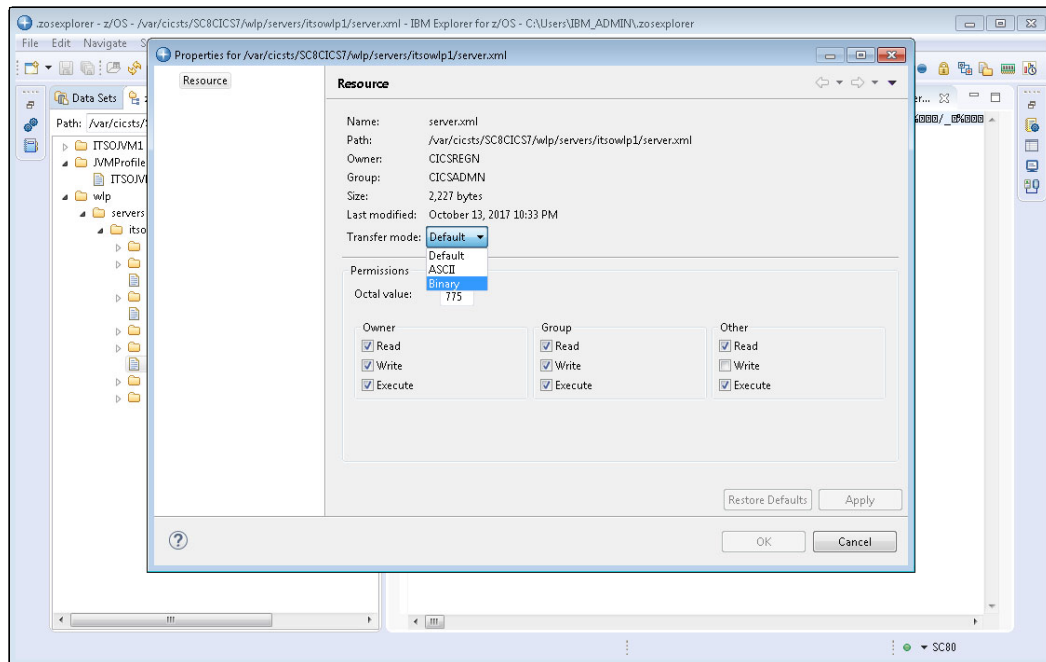


Figure 1-8 File properties pop-up window

UNIX tools

UNIX System Services provide a set of standard UNIX tools, such as more, vi, and cat for viewing and editing zFS files. These tools can be configured to manage tagged ASCII files that use the Enhanced ASCII function in UNIX System Services by setting the `_BPXX_AUTOCVT=ALL` environment variable in the user's shell.

1.3 Setting up a Liberty JVM Server

We are now ready to create and start our JVM server. In the following sections, we describe the process that is used to create a JVM profile and install the JVMSERVER resource definition into CICS.

1.3.1 JVM profile

The JVM profile is a zFS configuration file that lists the variables and system properties that are used by CICS when starting a JVM server. Some of the options are specific to, and others are standard for, the JVM runtime environment. For example, the JVM profile controls the initial size of the JVM storage heap and how far it can expand. The profile can also define the destinations for messages and dump output that is produced by the JVM. The JVM profile is named in the JVMPROFILE attribute in a JVMSERVER resource definition.

You can copy one of the sample JVM profiles that is provided by CICS in the USSHOME/JVMProfiles directory and customize it for your own application.

The sample JVM profiles in the USSHOME installation location are overwritten if you apply an APAR that includes changes to these files. To avoid losing your modifications, always copy the sample from USSHOME to a different location before adding or changing any options.

For more information about the properties that you can edit in a Liberty JVM server profile, see the [Options for JVMs in a CICS environment](#) in IBM Knowledge Center.

1.3.2 Tailoring the JVM profile

To begin tailoring your own JVM profile, complete the following steps:

1. Copy the sample JVM profile for a Liberty JVM server, as delivered with your CICS installation:

```
<USSHOME>/JVMProfiles/DFHWLP.jvmprofile
```

2. Copy the profile to your JVMPROFILEDIR directory as defined in the SIT. We used the following location:

```
/var/cicsts/SC8CICS7/JVMProfiles/ITS0JVM1.jvmprofile
```

You then must modify the options that are described next.

Java home directory

The JAVA_HOME directory specifies the installation location for the IBM Java SDK for z/OS. This location contains subdirectories and Java archive files that are required for Java support.

The supplied sample JVM profiles contain a path that was generated by the JAVADIR parameter in the DFHISTAR CICS installation job. The default for the JAVADIR parameter is /usr/lpp/java/J8.0/, which is the default installation location for the IBM 64-bit SDK for z/OS, Java Technology Edition.

Working directory

WORK_DIR is the working directory in zFS that the CICS region uses for activities that are related to the JVM server. The CICS JVM server uses this directory for configuration and output. A period (.) is defined in the supplied JVM profiles and indicates that the home directory of the CICS region user ID is to be used as the working directory. If the directory does not exist or if WORK_DIR is omitted, /tmp is used as the zFS directory name.

You can also specify your own zFS directory for the working directory. In this situation, you must also ensure that the relevant directory is created on zFS and that access permissions are granted to the correct CICS regions.

We specified the following pattern for our WORK_DIR under which CICS dynamically creates subdirectories for each region and JVM server:

```
WORK_DIR=/var/cicsts/
```

Tip: Always create the working directory in a different mount point than your CICS USSHOME directory because CICS requires read/write access to the working directory. The USSHOME should be mounted read-only because it is part of the product installation.

Output files

Output from Java applications that are running in a Liberty JVM server can be written to zFS or to JES. The destination for the logs is defined by using the STDOUT, STDERR, and JVMTRACE options in the JVM profile.

Standard output

The following standard output options are available:

- ▶ **STDERR**

This option specifies the location to which the stderr stream is directed from the JVM server. It often contains useful output, such as Java exception messages. By default, files are placed in the WORK_DIR/<applid>/<jvmserver> directory in zFS; however, it can be redirected to the JES log. If you left STDERR undefined, the output is written to the following default location in your working directory:

```
/var/cicsts/SC8CICS7/ITS0JVM1/Dyyyyymmdd.Thhmmss.dfhhjvmerr
```

- ▶ **STDOUT**

This option specifies the location to which the stdout stream is directed from the JVM server. By default, files are placed in the WORK_DIR/<applid>/<jvmserver> directory in zFS. Although CICS JVM servers do not log output to this file, it can be used by Java applications or other components. If you left STDOUT undefined, the output is written to the following location in your working directory:

```
/var/cicsts/SC8CICS7/ITS0JVM1/Dyyyyymmdd.Thhmmss.dfhhjvmout
```

- ▶ **JVMTRACE**

This option specifies the name of the zFS file or JES DD to which Java tracing is written during the operation of a JVM server. If you do not set a value for this option, CICS automatically creates unique trace files for each JVM server, although it can also be redirected to the JES log. This option is a CICS Java trace and the amount of tracing is controlled by setting the trace levels for the SJ and AP domains by using the CETR transaction:

```
/var/cicsts/SC8CICS7/ITS0JVM1/Dyyyyymmdd.Thhmmss.dfhhjvmtrc
```

- ▶ **Liberty messages.log**

This option contains all messages that are written out from the Liberty server. It also contains information, such as the message time stamp and the ID of the thread that wrote the message. In our scenario, the file is in zFS in the following location:

```
/var/cicsts/SC8CICS7/wlp/servers/itsowlp1/logs/messages.log
```

JVM server stdout and stderr streams can be redirected to JES instead of zFS, which enables the JVM server logs to be kept with the CICS output. For example, to log output to a specific JES DD file, you must specify the following options for the STDOUT and STDERR in the JVM profile:

```
STDOUT=//DD:JVMOUT
STDERR=//DD:JVMERR
```

We added the following JCL DD cards to our production CICS region to match the DD destination that was specified in the profile. The parameter LRECL is used to avoid excessive message wrapping and limits line length to 1024 characters:

```
// JVMOUT   DD SYSOUT=*,LRECL=1024
// JVMERR   DD SYSOUT=*,LRECL=1024
// JVMTRACE DD SYSOUT=*,LRECL=1024
```

For more information about how to manage logging output from the JVM server, see Chapter 8, “Logging and monitoring” on page 221.

Maximum number of zFS log or trace files

The LOG_FILES_MAX value in the JVM server profile specifies the number of old log JVM server stdout, stderr, and dfhjvmtrc files are kept on zFS. A default setting of 0 ensures that all old versions of the log file are retained. This value can be modified to specify how many old log files you want to remain on the file system, as shown in the following example:

```
LOG_FILES_MAX=<n>
```

If STDOUT, STDERR, and JVMTRACE use the default scheme, or if they are customized to include the &DATE;.&TIME; pattern, only the newest instance of each log type is kept on the system. If your customization does not include any variables that make the output unique, the files are appended to, and no requirement is necessary for deletion. This clean-up function does not apply if the output variables were customized to route output to JES.

Time zone

The time zone (TZ) environment variable specifies the local time of a system. You can set this variable for a JVM server by adding it to the JVM profile.

When setting the time zone for a JVM server, consider the following points:

- ▶ The TZ variable in your JVM profile must match your local MVS system offset from Greenwich mean time (GMT).
- ▶ If you do not set the TZ variable, the system defaults to Coordinated Universal Time (UTC).
- ▶ Customized time zones are not supported and result in failover to UTC or a mixed time zone output in the JVMTRACE file.
- ▶ If you see LOCALTIME as the time zone string, an inconsistency occurs in your configuration. This inconsistency can be between your local MVS time and the TZ you are setting, or between your local MVS time and your default setting in the JVM profile. The output is in mixed time zones, although each entry is correct.
- ▶ The short form of Portable Operating System Interface (POSIX) TZ can be used and reduces the chances of input errors and uses the following format:

```
TZ=CET-1CEST
```

The long form uses the following format:

```
TZ=CET-1CEST,M3.5.0,M10.5.0
```

1.3.3 Liberty specific options

The following variables are specific to a Liberty JVM server environment:

- **WLP_INSTALL_DIR**

This variable specifies the installation directory of the Liberty server product. The Liberty files are installed in the CICS USSHOME directory in a subdirectory that is named wlp. The default installation directory is /usr/lpp/cicsts/cicsts54/wlp. Always use the &USSHOME; symbol to set the correct file path and append the wlp directory, as shown in the following example:

```
WLP_INSTALL_DIR=&USSHOME;/wlp
```

- **WLP_USER_DIR**

This variable specifies the directory that contains the configuration files for the Liberty JVM server. This environment variable is optional. If you do not specify this variable, CICS defaults to the following subdirectory of the JVM server working directory:

```
WLP_USER_DIR=./&APPLID;/&JVMSERVER;/wlp/usr.
```

In our configuration we used:

```
WLP_USER_DIR=/var/cicsts/&APPLID;/wlp
```

- **WLP_OUTPUT_DIR**

This optional variable specifies the directory that contains output files for the Liberty server. By default, Liberty stores logs, the work area, and configuration files for the server in a directory that is named after the server. If you do not specify this variable, CICS defaults to the following subdirectory of the JVM server working directory:

```
WLP_OUTPUT_DIR=./&APPLID;/&JVMSERVER;/wlp/usr/servers
```

In our configuration, this variable defaulted to the following directory:

```
WLP_USER_DIR=/var/cicsts/&APPLID;/wlp/usr/servers
```

The following system properties are specific to the Liberty JVM server environment:

- **Encoding**

A Liberty JVM server requires that JVM encoding be set to ASCII rather than the usual EBCDIC default for CICS. This setting is controlled by using the file.encoding JVM system property. You must set your encoding as shown in the following example to start a Liberty JVM server:

```
-Dfile.encoding=ISO-8859-1
```

- **Autoconfigure**

Enable autoconfigure for at least the first startup of the Liberty JVM server in CICS, which builds the CICS Liberty server environment. This process requires setting the following system property:

```
-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true
```

This property instructs CICS to build the Liberty server and configuration file (server.xml) during startup. The default JVM profile supplies this line and you must remove the '#' comment and set the value to true.

After you start your JVM server and it is working correctly, you can continue to leave autoconfigure enabled or you can disable autoconfigure and perform all further changes manually to server.xml.

Tip: If you are using autoconfigure, the HTTP ports and host are configured by using system properties; therefore, they are not modified directly in the `server.xml` file.

► HTTP/HTTPS ports

Change your default ports so that they do not conflict with any port usage. Delete the '#' before the following two lines and change '9080' and '9443' to valid free ports:

```
-Dcom.ibm.cics.jvmserver.wlp.server.http.port=57080
-Dcom.ibm.cics.jvmserver.wlp.server.https.port=57443
```

You can also add the `com.ibm.cics.jvmserver.wlp.server.host` property to restrict the HTTP endpoint to bind to a specific host:

```
-Dcom.ibm.cics.jvmserver.wlp.server.host=wtsc80
```

For more information about configuring SSL, see Chapter 6, "Configuring Transport Layer Security support" on page 141.

► Our sample JVM profile

The result of editing `ITS0JVM1.jvmprofile` is shown in Example 1-3.

Example 1-3 JVM profile

```
WORK_DIR=/var/cicsts/
WLP_INSTALL_DIR=&USSHOME;/wlp
WLP_USER_DIR=/var/cicsts/&APPLID;/wlp
TZ=EST5EDT
LOG_FILES_MAX=5
-Xms128
-Xmx256
-Xms0128-Xgcpolicy:gencon
-Xscmx128-Xshareclasses:name=cicsts540%g,groupAccess,nonfatal
-Dcom.ibm.tools.attach.enable=no

-Dcom.ibm.ws.logging.max.files=5
-Dfile.encoding=ISO-8859-1

# Autoconfigure options
-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true
-Dcom.ibm.cics.jvmserver.wlp.server.name=itsowlp
-Dcom.ibm.cics.jvmserver.wlp.server.http.port=57080
-Dcom.ibm.cics.jvmserver.wlp.server.https.port=57443
-Dcom.ibm.cics.jvmserver.wlp.server.host=wtsc80
```

1.3.4 IBM Language Environment runtime options

The IBM Language Environment® runtime options module is specified in the JVMSERVER resource definition and defines the runtime options for the Language Environment enclave that is used by the JVM server. DFHAXRO is the supplied program that provides a set of default values. The source for DFHAXRO is in the h1q.SDFHSAMP library and can be used to modify the Language Environment storage options that are used by the JVM server, if required.

1.3.5 Creating a JVMSERVER resource

The CICS JVMSERVER resource definition must be created next. This definition is used to set the JVM profile location and to control the lifecycle of the JVM server within the CICS runtime environment.

We used the values that are listed in Table 1-2.

Table 1-2 Values for the defining the JVMSERVER

Attribute	Default	Value	Description
JVMSERVER	DFHWLP	ITSOJVM1	JVM server resource name
Group	DFH\$WLP	LIBERTY	CSD group name
Jvmprofile	DFHWLP	ITSOJVM1	Prefix for .jvmprofile in zFS
Liberty server Name	defaultServer	itsowlp1	Liberty server name as set by property com.ibm.cics.jvmserver.wlp.server.name

You can define the JVMSERVER resource by using the following tools:

- ▶ Resource definition online (RDO) by using the CEDA transaction
- ▶ Submitting offline batch jobs by using the DFHCSDUP batch utility program
- ▶ CICS Explorer

We used the CEDA transaction to define the JVM server. Complete the following steps:

1. Using the CEDA transaction, copy the sample JVMSERVER DFHWLP resource definition from the sample CSD group DFH\$WLP to a new CSD group of your choice to enable it to be modified.
2. Create a JVMSERVER definition by using CEDA with your user-specific changes, as shown in Figure 1-9.

```
QVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA ALTER JVMSERVER( ITSOJVM1 )
  JVMSERVER      : ITSOJVM1
  Group          : LIBERTY
  DESCRIPTION    ==> PRODUCTION JVM SERVER WITH LIBERTY
  Status         ==> Enabled           Enabled | Disabled
  Jvmprofile     ==> ITSOJVM1           (Mixed Case)
  Lerunopts      ==> DFHAXRO
  Threadlimit    ==> 050                1-256
  DEFINITION SIGNATURE
  DEFINETIME     : 10/06/17 14:22:06
  CHANGETIME     : 10/06/17 14:22:15
  CHANGEUSRID    : CICSUSER
  CHANGEAGENT    : CSDAPI              CSDAPI | CSDBatch
  CHANGEAGREL    : 0710

                                           SYSID=SC87 APPLID=SC8CICS7

PF 1 HELP 2 COM 3 END                6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 1-9 CEDA JVMSERVER definition.

3. Install and enable the JVMSERVER resource into your CICS region that uses CEDA by using the **CEDA EXPAND GROUP(LIBERTY)** command, as shown in Figure 1-10.

```

EX G(liberty)
ENTER COMMANDS
NAME      TYPE      GROUP      LAST CHANGE
ITSOJVM1  JVMSERVER  LIBERTY    install
10/06/17  14:22:15

RESULTS: 1 TO 1 OF 1
SYSID=SC87 APPLID=SC8CICS7
TIME: 16.44.38 DATE: 10/06/17
PF 1 HELP 2 SIG 3 END 4 TOP 5 BOT 6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL

```

Figure 1-10 Install JVMSERVER using CEDA

4. Check the status of the JVM server by using CEMT INQ JVMSERVER transaction, as shown in Figure 1-11.

```

INQ JVM
RESULT - OVERTYPE TO MODIFY
Jvmserver(ITS0JVM1)
Enablestatus( Enabled )
Purgetype( )
Prfile(ITS0JVM1)
Lerunopts (DFHAXR0)
Threadcount(012)
Threadlimit( 050 )
Currentheap(49298552)
Iniheap(128M)
Maxheap(256M)
Gcpolicy(-Xgcpolicy:gencon)
Occupancy(43286368)
Pid(0033621804)
Profiledir(/var/cicsts/SC8CICS7/JVMProfiles)
Installtime(10/06/17 16:16:51)
Installusrid(CICSUSER)
Installagent(Grplist)
+ Definesource(LIBERTY)

SYSID=SC87 APPLID=SC8CICS7
TIME: 16.32.27 DATE: 10/06/17
PF 1 HELP 2 HEX 3 END 5 VAR 7 SBH 8 SFH 10 SB 11 SF

```

Figure 1-11 CEMT INQUIRE JVMSERVER

If the status shows Enabled, your JVM server environment should be running. You completed your first step and successfully validated the Java environment on CICS. If it fails to enable, check the CSMT log, SYSOUT log, and stderr log in the zFS working directory for more information about any errors.

Restarting the JVM server

Although changes to the Liberty server configuration are dynamic, any changes to the JVM profile require restarting the JVM server. To stop the JVM server, you can use the **CEMT SET JVMSERVER DISABLED** command. The default action is to phase-out the JVM server; however, the following actions are available if shutdown does not proceed as wanted:

- ▶ **PHASEOUT** is the default action and shuts down the JVM server. All running tasks continue until completion, but no new work is accepted by the JVM server. When all of the tasks are finished, the JVMSERVER resource enters the DISABLED state.
- ▶ The **PURGE** action purges tasks that are running in the specified JVM server. Any threads that are running in the JVM are instructed to stop. CICS purges tasks only when system and data integrity can be maintained. If the JVMSERVER resource remains in the BEING DISABLED state, some tasks cannot be purged.
- ▶ **FORCEPURGE** forces purge tasks that are running in the specified JVM server. Any threads that are running in the JVM are instructed to stop. If the JVMSERVER resource remains in the BEING DISABLED state, some tasks could not be force purged. Data integrity is not ensured.
- ▶ **KILL** ends tasks that are running in the specified JVM server. Any threads that are running in the JVM are stopped. The JVMSERVER resource enters the DISABLED state and all work is ended.

Note: In CICS TS V5.4, the **SET TASK PURGE** command is now supported for ending tasks that are running in a JVM server. This command is also available for CICS TS V5.3 with the fix for APAR PI77502.

1.4 Tailoring server.xml

In this section, we describe how you can manually configure your `server.xml` file and set up the XML elements within it. Performing a manual configuration is important in a production environment in which you might want to more tightly control the server configuration.

One of the major advantages of Liberty is its composability, which is based on features. You can add all the functionality that you need for your specific set of applications by choosing the features that they require. Being composable allows you to keep the server lightweight because you add only the features that you need.

1.4.1 Adding Liberty features

You can add various features in the `<featureManager>` list in `server.xml`. The following key features were used in our initial configuration (see Example 1-4 on page 19):

- ▶ The CICS feature `cicsts:core-1.0` is the core integration feature that provides the runtime integration, JCICS API, and transaction support for JTA. It should always be configured.
- ▶ The `ssl-1.0` feature enables Secure Sockets Layer (SSL) support for the HTTP listeners by using Java keystores or RACF key rings.
- ▶ The `jsp-2.3` feature enables support for servlet and JavaServer Pages (JSP) applications at the Java EE 7 web profile standard. This feature is required by Dynamic Web projects (WAR files) and OSGi Application Projects that contain OSGi bundle projects with web support and also enables the `servlet-3.1` feature.

- The `jaxrs-2.0` feature enables support for RESTful Java applications developed using the JAXRS APIs. This feature is required by our `restapp` and `restapp` extensions projects.

Example 1-4 Server.xml after adding new features

```
<featureManager>
  <feature>cicsts:core-1.0</feature>
  <feature>ssl-1.0</feature>
  <feature>jsp-2.3</feature>
  <feature>jaxrs-2.0</feature>
</featureManager>
```

CICS TS V5.4 and V5.3 support all of the features from the Java EE 7 full profile that are provided by WebSphere Liberty. This configuration enables Java EE web applications to be deployed into a Liberty JVM server.

To verify that the features are installed, check the message `CWWKF0012I` in the `MSGLOG` file, as shown in Example 1-5.

Example 1-5 Liberty messages.log installed features

```
CWWKF0012I: The server installed the following features: [jsp-2.3, servlet-3.1,
ssl-1.0, jndi-1.0, appSecurity-2.0, jaxrs-2.0, jaxrsClient-2.0, el-3.0,
blueprint-1.0, cicsts:core-1.0, json-1.0, distributedMap-1.0, wab-1.0].
```

We can see each of the features from our configuration list, along with several other features, such as `jndi-1.0`, `appSecurity-2.0`, `jaxrsClient-2.0`, `el-3.0`, `distributedMap-1.0`, and `wab-1.0`, that were included as part of other features.

1.4.2 Include files

If configuration information is in an external `.xml` file, you can use the `<include>` element to add the configuration information to the `server.xml` file. For example, as described in Chapter 5, “Connecting to IBM MQ by using JMS” on page 113, we used an include file that is named `jms.xml` with IBM MQ JMS-specific definitions for our JMS application. The following example shows how the include element in our `server.xml` appeared:

```
<include location="/var/cicsts/SC8CICS7/wlp/servers/itsowlp/jms.xml"
optional="true"/>
```

You can configure the `onConflict` attribute on the `<include>` element file to manage conflict between the `server.xml` file and the include file when duplicate values are defined. This attribute can be configured to one of the following values:

- `merge`: Allows Liberty to merge conflicting elements (default setting)
- `replace`: Causes the include elements to override `server.xml`
- `ignore`: Ignores elements in the include that conflict with the `server.xml`

1.4.3 Configuring the HTTP and HTTPS endpoints

If HTTP and HTTPS ports are not defined, they default to 9080 (HTTP) and 9443 (HTTPS). If you want to change the port or IP address that is used by the Liberty HTTP listener, update the `<httpEndpoint>` element with the host name and port numbers that you require, as shown in Example 1-6 on page 20.

Example 1-6 HTTP and HTTPS port definitions in server.xml

```
<httpEndpoint id="defaultHttpEndpoint"
  host="wtsc80"
  httpPort="57080"
  httpsPort="57443" />
```

Use a port number that is not in use anywhere else; for example, by a TCPIP SERVICE in CICS. HTTPS is available only if SSL is configured.

If a multi-homed TCP/IP configuration with multiple IP addresses defined is used, you can configure a Liberty server to listen on different IP endpoints by creating multiple `<httpEndpoint>` elements.

How we defined three listeners on port 57080 for each of the three host names that are defined on our system (wtsc80, wtsc80oe, and localhost) is shown in Example 1-7. You can also control specific functionality for each endpoint because SSL is disabled by setting `httpsPort="-1"` in the `defaultHttpEndpoint3`, as shown in Example 1-7.

Example 1-7 Multiple HTTP endpoints in server.xml

```
<httpEndpoint id="defaultHttpEndpoint"
  host="wtsc80"
  accessLoggingRef="accessLogging"
  httpOptionsRef="httpoptions"
  httpPort="57080"
  httpsPort="57443" />

<httpEndpoint id="defaultHttpEndpoint2"
  host="wtsc80oe"
  httpOptionsRef="httpoptions"
  httpPort="57080"
  httpsPort="57443" />

<httpEndpoint id="defaultHttpEndpoint3"
  host="localhost"
  httpOptionsRef="httpoptions2"
  httpPort="57080"
  httpsPort="-1"/>
```

To verify the status of the HTTP listeners, use the **NETSTAT** command to verify the active listeners in use by a specific job, as shown in Example 1-8.

Example 1-8 Active listeners for job SC8CICS7

```
/D TCPIP,,NETSTAT,ALLCONN,CLIENT=SC8CICS7
MVS TCP/IP NETSTAT CS V2R3          TCPIP Name: TCPIP          11:30:15
User Id Conn      Local Socket          Foreign Socket          State
-----
SC8CICS7 00007C7B 127.0.0.1..57080        0.0.0.0..0             Listen
SC8CICS7 00007C79 9.76.61.131..57080      0.0.0.0..0             Listen
SC8CICS7 00007D4D 9.76.61.132..57080      0.0.0.0..0             Listen
```

1.4.4 CICS bundle deployed applications

If you want to deploy Liberty applications in a CICS bundle, the `server.xml` file must include the entry that is shown in Example 1-9.

Example 1-9 Setting the include location for CICS bundles

```
<!-- CICS Bundle Installed Applications -->  
<include location="${server.output.dir}/installedApps.xml"/>
```

The included `installedApps.xml` file is used to define the CICS bundle-deployed applications when they are enabled in CICS.

1.4.5 Default web application

The CICS default web application `CICSDefaultApp` is a built-in web application that can be used to validate that the Liberty JVM server started with the correct configuration. Although it is not suited to a production system, it is useful for verifying the configuration.

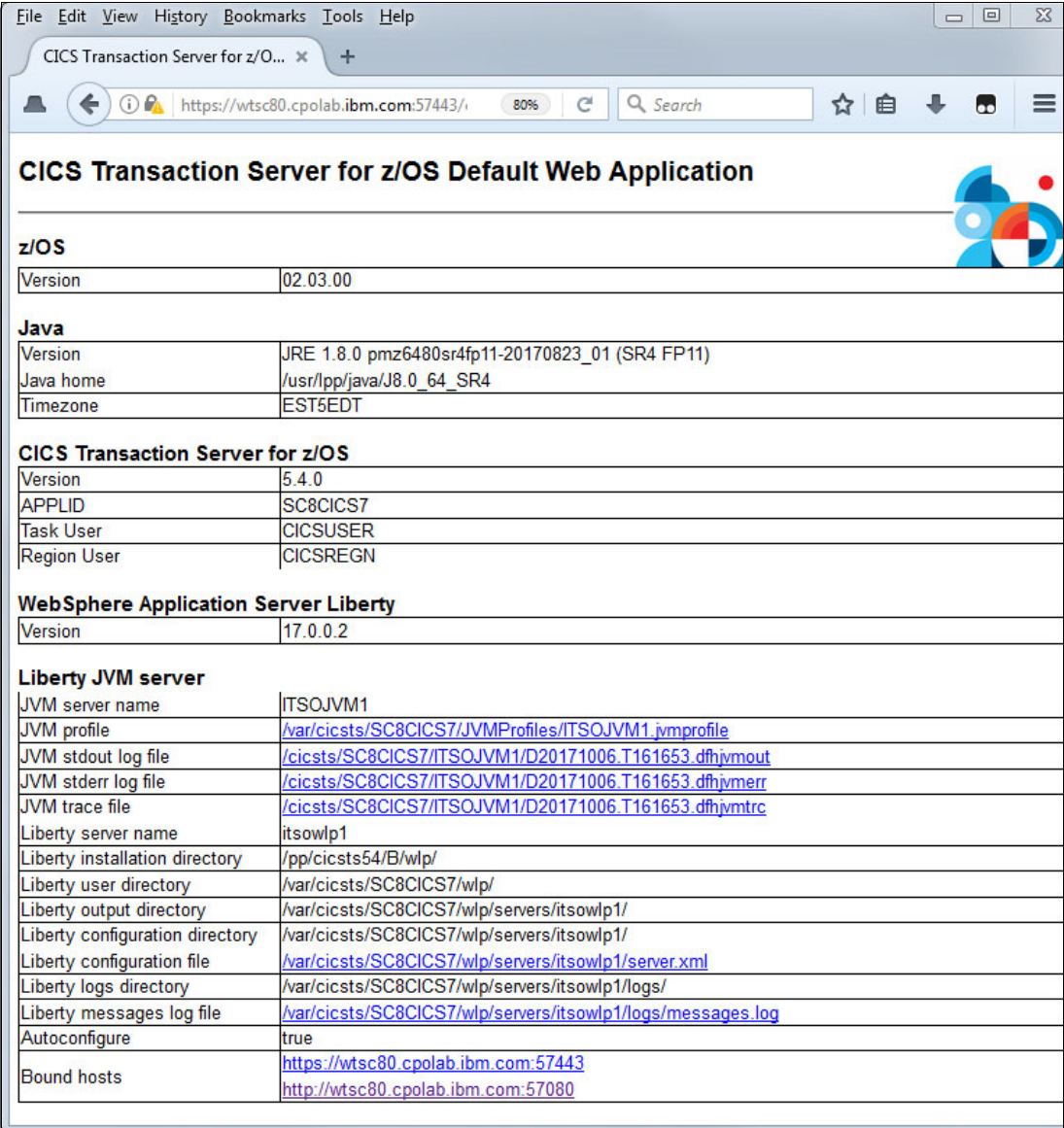
If `autoconfigure` is used, add the statement that is shown in Example 1-10 to your JVM profile file. A new feature (`cicsts:defaultApp-1.0`) is added to your `server.xml` file. If `autoconfigure` is not used, manually add the feature to `server.xml`.

Example 1-10 Enabling the default web application

```
-Dcom.ibm.cics.jvmserver.wlp.defaultapp=true
```

You can run the application by using the following URL, which returns a web page that is similar to the page that is shown in Figure 1-12:

`http://<server>:<port>/com.ibm.cics.wlp.defaultapp/`



The screenshot shows a web browser window with the address bar displaying `https://wtsc80.cpolab.ibm.com:57443/`. The page title is "CICS Transaction Server for z/OS Default Web Application". The page content is organized into several sections, each with a table of properties:

- z/OS**

Version	02.03.00
---------	----------
- Java**

Version	JRE 1.8.0 pmz6480sr4fp11-20170823_01 (SR4 FP11)
Java home	/usr/lpp/java/J8.0_64_SR4
Timezone	EST5EDT
- CICS Transaction Server for z/OS**

Version	5.4.0
APPLID	SC8CICS7
Task User	CICSUSER
Region User	CICSREGN
- WebSphere Application Server Liberty**

Version	17.0.0.2
---------	----------
- Liberty JVM server**

JVM server name	ITSOJVM1
JVM profile	/var/cicsts/SC8CICS7/JVMProfiles/ITSOJVM1.jvmprofile
JVM stdout log file	/cicsts/SC8CICS7/ITSOJVM1/D20171006.T161653.dfthjvmout
JVM stderr log file	/cicsts/SC8CICS7/ITSOJVM1/D20171006.T161653.dfthjvmerr
JVM trace file	/cicsts/SC8CICS7/ITSOJVM1/D20171006.T161653.dfthjvmtrc
Liberty server name	itsowlp1
Liberty installation directory	/pp/cicsts54/B/wlp/
Liberty user directory	/var/cicsts/SC8CICS7/wlp/
Liberty output directory	/var/cicsts/SC8CICS7/wlp/servers/itsowlp1/
Liberty configuration directory	/var/cicsts/SC8CICS7/wlp/servers/itsowlp1/
Liberty configuration file	/var/cicsts/SC8CICS7/wlp/servers/itsowlp1/server.xml
Liberty logs directory	/var/cicsts/SC8CICS7/wlp/servers/itsowlp1/logs/
Liberty messages log file	/var/cicsts/SC8CICS7/wlp/servers/itsowlp1/logs/messages.log
Autoconfigure	true
Bound hosts	https://wtsc80.cpolab.ibm.com:57443 http://wtsc80.cpolab.ibm.com:57080

Figure 1-12 Default web application output

1.4.6 Liberty transaction log files

The Java Transaction API (JTA) can be used in a Java EE application to coordinate transactional updates to third-party XA resource managers, such as connections to DB2 that use JDBC type 4 connectivity. In this scenario, the Liberty transaction manager is the transaction coordinator and stores its recovery information in the JTA transaction logon zFS.

On restart, the Liberty transaction manager starts XA transaction recovery when the JVM server initialization is complete. This process ensures that any transactions in the remote resource managers are recovered together with any in-doubt units-of-work in CICS.

The integrity of the CICS system log and the Liberty transaction log is critical in enabling CICS to perform successful recovery if the Liberty JVM server fails while recoverable updates are still in-flight. The default location for the transaction logs is `${WLP_USER_DIR}/tranlog/`. This location can be overridden by modifying the `<transaction>` element in `server.xml` as shown in the following example:

```
<transaction transactionLogDirectory="${server.config.dir}/tranlog/" />
```

These logs files are used to recover JVM status at start (see Figure 1-13). To preserve system consistency, they should not be deleted.

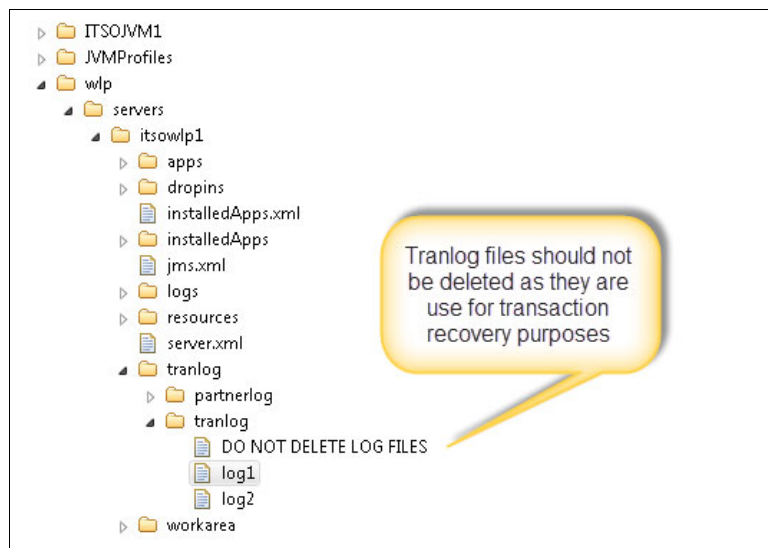


Figure 1-13 Liberty JTA transaction log directory

1.4.7 Sample server.xml file

Example 1-11 shows what our server.xml file looked like when the configuration was complete.

Example 1-11 Sample server.xml with the updates

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="CICS Liberty profile sample configuration">
  <featureManager>
    <feature>cicsts:core-1.0</feature>
    <feature>ssl-1.0</feature>
    <feature>jsp-2.3</feature>
    <feature>jaxrs-2.0</feature>
    <feature>cicsts:defaultApp-1.0</feature>
  </featureManager>

  <safRegistry enableFailover="false" id="saf"/>

  <!-- HTTP End Point -->
  <httpEndpoint id="defaultHttpEndpoint"
    accessLoggingRef="accessLogging"
    host="wtsc80"
    httpPort="57080"
    httpsPort="57443" />

  <httpAccessLogging id="accessLogging"
    filepath="${server.output.dir}/logs/http_access.log"
    logFormat="%t %a %i %r %s %u %D %B" />

  <httpOptions id="httpoptions"
    keepAliveEnabled="false"
    maxKeepAliveRequests="1" />

  <!-- CICS Bundle Installed Applications -->
  <include location="${server.output.dir}/installedApps.xml"/>

  <config monitorInterval="5s"
    updateTrigger="polled" />

  <!-- Monitoring of application updates -->
  <applicationMonitor
    dropins="dropins"
    dropinsEnabled="true"
    pollingRate="5s"
    updateTrigger="disabled" />

  <!-- Enable specific applications by includes -->
  <include
    location="/var/cicsts/SC8CICS7/wlp/servers/itsowlp/jms.xml"
    optional="true" />

  <executor id="allowCICSconfigure" maxThreads="50"/>
</server>
```

1.4.8 Welcome page

A quick way to check your Liberty JVM server status is to use the home page that is created by the Liberty server. This page is a simple welcome page, which is displayed by Liberty when a browser attempts to access its HTTP or HTTPS port without a URL path. You can access this page by at the URL `<hostname>:<port>`.

In our example, we used `http://wtsc80.cpolab.ibm.com:57080/`, which displayed the page that is shown in Figure 1-14.

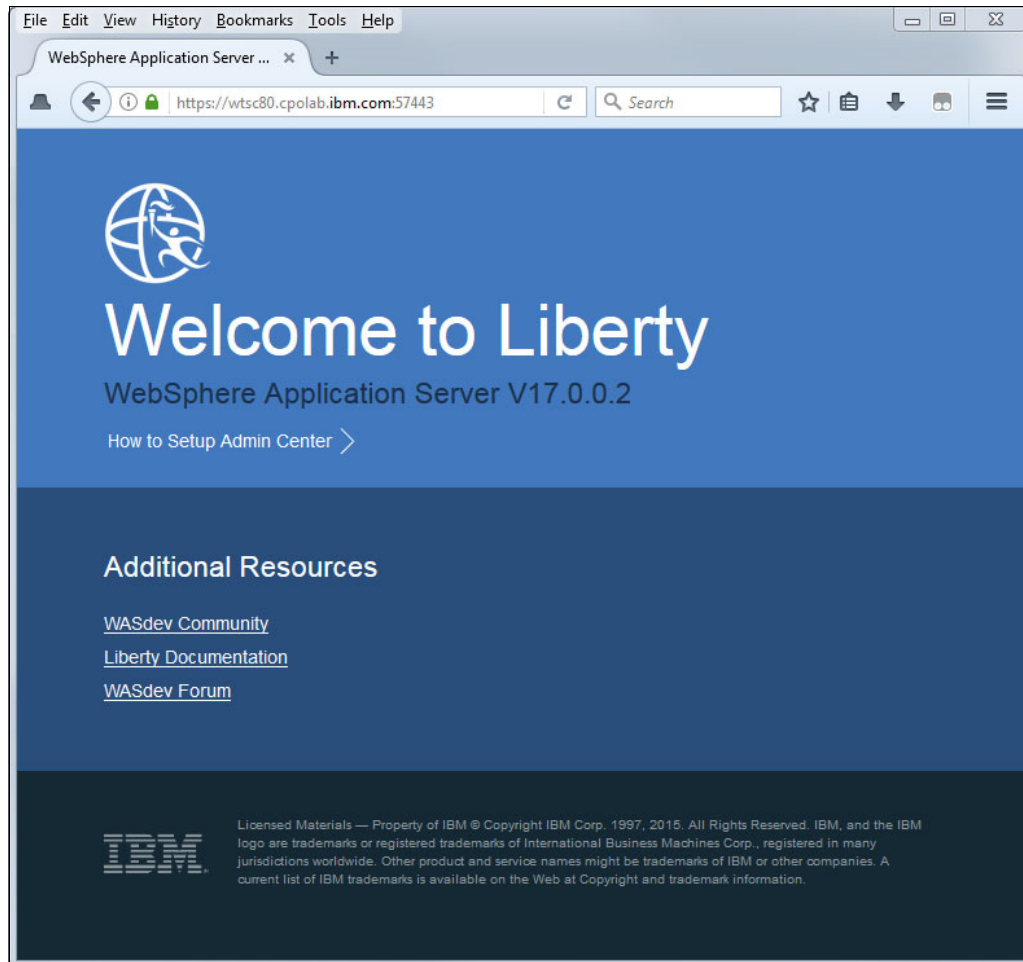


Figure 1-14 Liberty welcome page

If you want to disable this welcome page in a production environment, use the following parameter:

```
<httpDispatcher enableWelcomePage="false" />
```

If you want to give a customized message to user in this situation, you can add another parameter to `<httpDispatcher>` element, as shown in the following example:

```
<httpDispatcher enableWelcomePage="false"
  appOrContextRootMissingMessage="Page unavailable" />
```




Deploying a web application

In this chapter, we describe the three options that are available to deploy a web application into a CICS Liberty JVM server and the key differences between these options.

We used the CICSDev sample `restapp`, which is a simple RESTful web application that is deployed by using an Eclipse dynamic web project as a WAR archive and [is available at the GitHub website](#). Similar considerations and procedures apply to any web application that you want to build from source and deploy to a CICS Liberty server.

This chapter includes the following topics:

- ▶ 2.1, “Building the `restapp` sample ” on page 28
- ▶ 2.2, “Deploying a web application to Liberty ” on page 32
- ▶ 2.3, “Advanced deployment options” on page 53

2.1 Building the restapp sample

In this section, we describe how to download the application from GitHub, import it into a CICS Explorer development environment, and build the .war file.

2.1.1 Obtaining the sample code

The source project for restapp is available at the [cicsdev GitHub website](#) (search for “cics-java-liberty-restapp”). The results for matching repositories are shown in Figure 2-1.

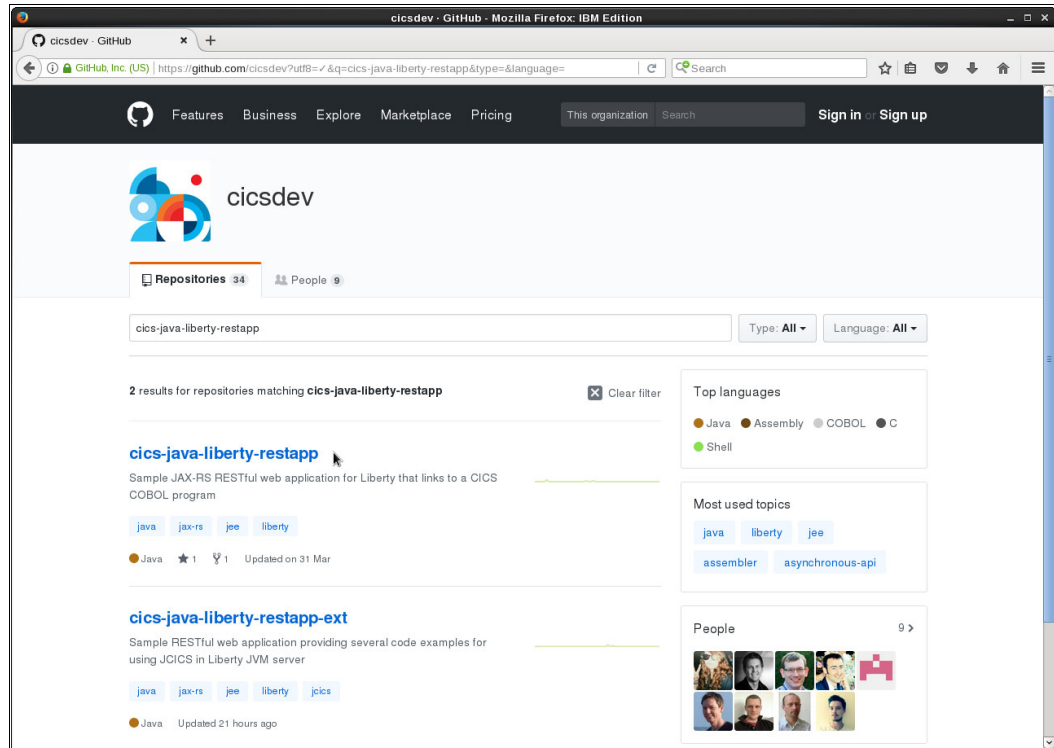


Figure 2-1 GitHub cicsdev repositories view

The following related RESTful web applications are available:

- ▶ **cics-java-liberty-restapp**

The restapp sample is a sample Java web application that provides RESTful APIs that uses JAX-RS. The following services are provided:

- **cicsinfo**: Includes no intrinsic CICS dependency, and might be on any Java EE server.
- **reverse**: Uses a CICS COBOL program to reverse an input string.

- ▶ **cics-java-liberty-restapp-ext**

The restapp-ext sample differs from the basic restapp because it consists of a suite of examples that use the JCICS API to access various CICS resources, and a specially annotated linkable Java class.

Complete the following steps to download restapp:

1. Select the **cics-java-liberty-restapp** repository, click **Clone or download** → **Download ZIP** (see Figure 2-2). The master branch of the repository is downloaded as `cics-java-liberty-restapp-master.zip` into your download directory.

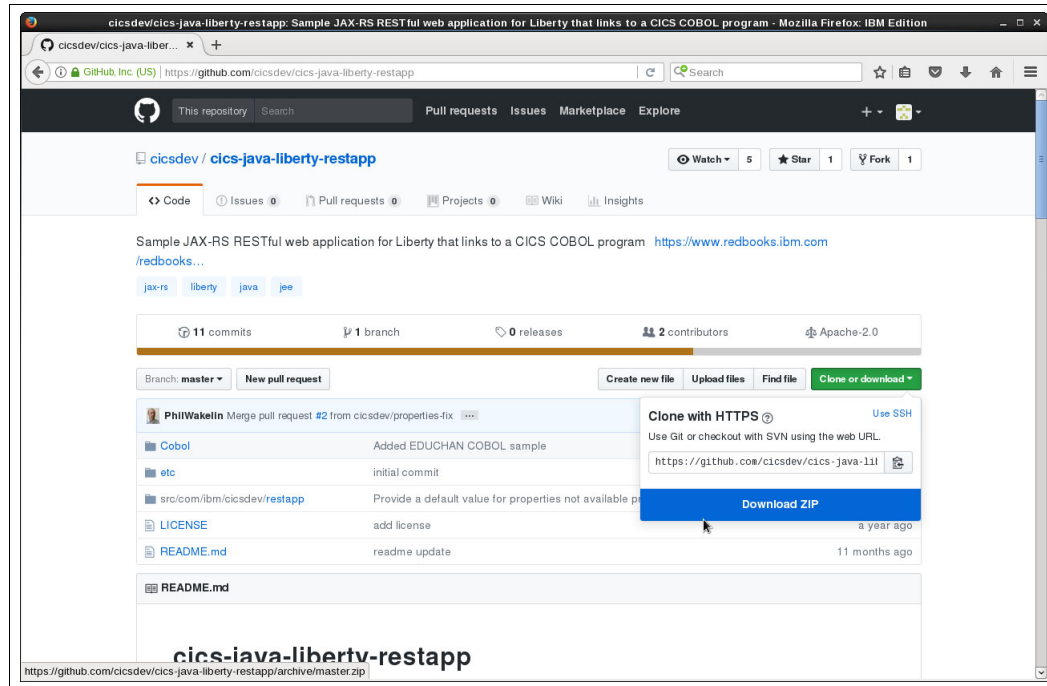


Figure 2-2 GitHub restapp repository download

2. Extract the downloaded repository .zip files to a suitable location on your workstation. The `/cics-java-liberty-restapp-master` directory is created, which includes the artifacts for the sample.

2.1.2 Creating the Eclipse projects

To build and deploy the sample application, you need an Eclipse development environment. We used CICS Explorer version 5.4 with the CICS SDK for Java EE and Liberty plug-in, which was installed by using into the IBM Explorer for z/OS Aqua V3.1. This installation is available at the [IBM Installation Manager website](#).

After your Eclipse development environment is installed, complete the following steps to create a dynamic web project and a CICS bundle project:

1. In CICS Explorer, switch to the Java EE perspective by clicking **Window** → **Perspective** → **Open Perspective** → **Other** → **Java EE**. Create a dynamic web project that is named `com.ibm.cicsdev.restapp` by using clicking **File** → **New** → **Dynamic Web Project**. Add the Project name `com.ibm.cicsdev.restapp`, as shown in Figure 2-3 on page 30.

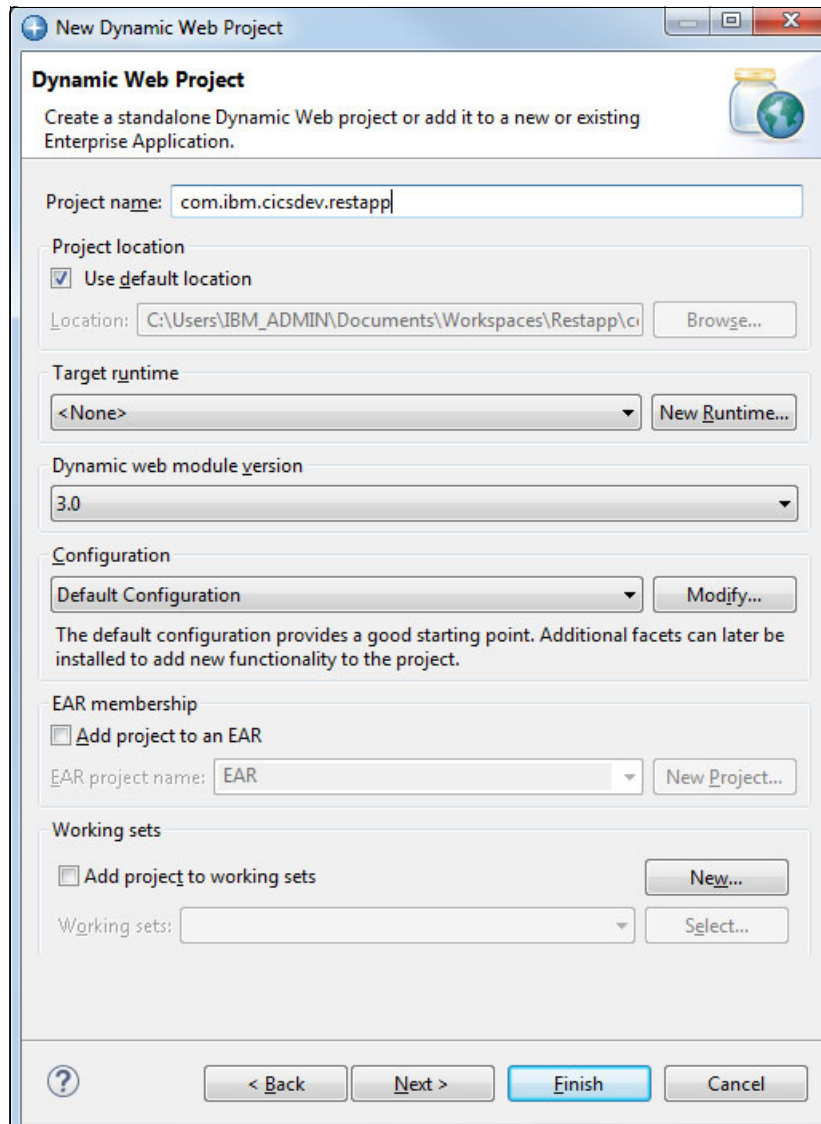


Figure 2-3 Creating dynamic web project for restapp

2. Click **Finish** to complete (the defaults that are shown in the next windows of the process are used). A project is created that is named `com.ibm.cicsdev.restapp`.

Note: A `web.xml` deployment descriptor is not created now. Such a descriptor is not essential, although one is needed to specify some advanced deployment options that are used in Chapter 7, “Securing web applications” on page 175.

3. Add the Java source code to the `src` folder in the Eclipse project. In the Windows OS, complete the following steps:
 - a. Open the `cics-java-liberty-restapp-master` folder. Select the **src/com** folder and drag this folder into the `src` folder in the Eclipse project.
 - b. Click **Copy files and folders** when prompted in the File and Folder Operation window (see Figure 2-4 on page 31). Click **OK** to finish.

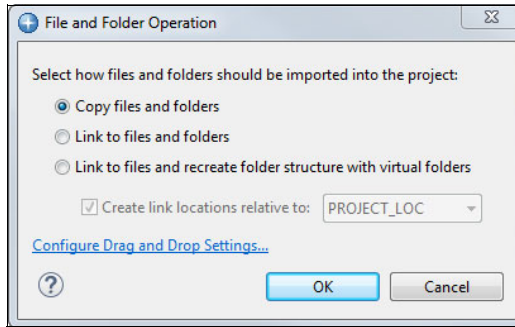


Figure 2-4 File and Folder Operation window

Two new packages, `com.ibm.cicsdev.restapp` and `com.ibm.cicsdev.restapp.bean`, are added to the `/src` folder, as shown in Figure 2-5. Build errors are flagged within the `com.ibm.cicsdev.restapp` package because the build path is incomplete at this stage.

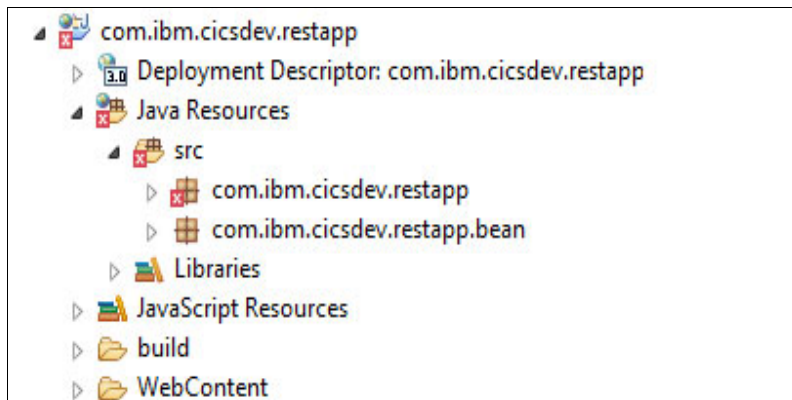


Figure 2-5 Eclipse project after import

4. Complete the following steps to add the Liberty JVM server libraries to the build path of your project:
 - a. Right-click the Eclipse **com.ibm.cicsdev.restapp** project and select **Build Path** → **Configure Build Path** in the Java Build Path window.
 - b. Select the **Libraries** tab and click **Add Library**.
 - c. Select library type **CICS with Java EE and Liberty** and click **Next**.
 - d. Select version **CICS TS 5.4** and click **Finish**. Then, click **OK** to complete the process.

The project build path is updated and the build errors in the project are removed as it is rebuilt.
5. Now we need to ensure that the web project is targeted to compile at a level that is compatible with the Java level that is being used on CICS. The compiler target level must be less than or equal to the CICS runtime Java level. We used the Java V8 SDK in CICS; therefore, the web application Java target level must be 1.8 or earlier.

A compatible level can be selected or checked in the Java project facet. Right-click the Eclipse **com.ibm.cicsdev.restapp** project and select **Properties**. Then, enter Project Facets. Click **Project Facets** and the configured facets are displayed (see Figure 2-6 on page 32). The Java Version should be equal or less than the Java level that is used in the Liberty JVM server (1.7 or 1.8). Correct this information if needed.

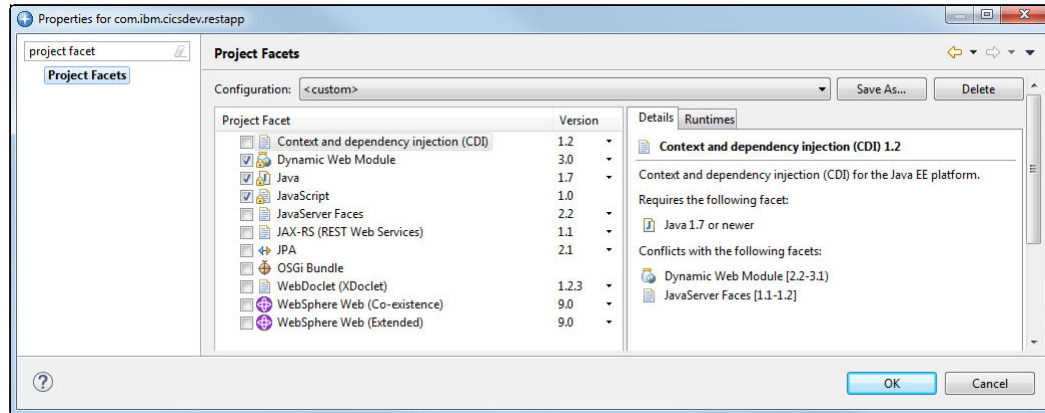


Figure 2-6 Project facets: Java version

2.2 Deploying a web application to Liberty

The following projects can be used to package Java EE web projects for deployment into Liberty:

- Dynamic web projects, for deployment as a WAR file archive
- Enterprise application projects, for deployment as an EAR file archive
- OSGi application projects, for deployment as an EBA file archive

In our example, we use dynamic web projects with the .war file archive type.

The following options are available for a Liberty JVM server to provide for the deployment of web applications (see Figure 2-7):

- By using the Liberty dropins directory
- As a Liberty application
- As a CICS bundle

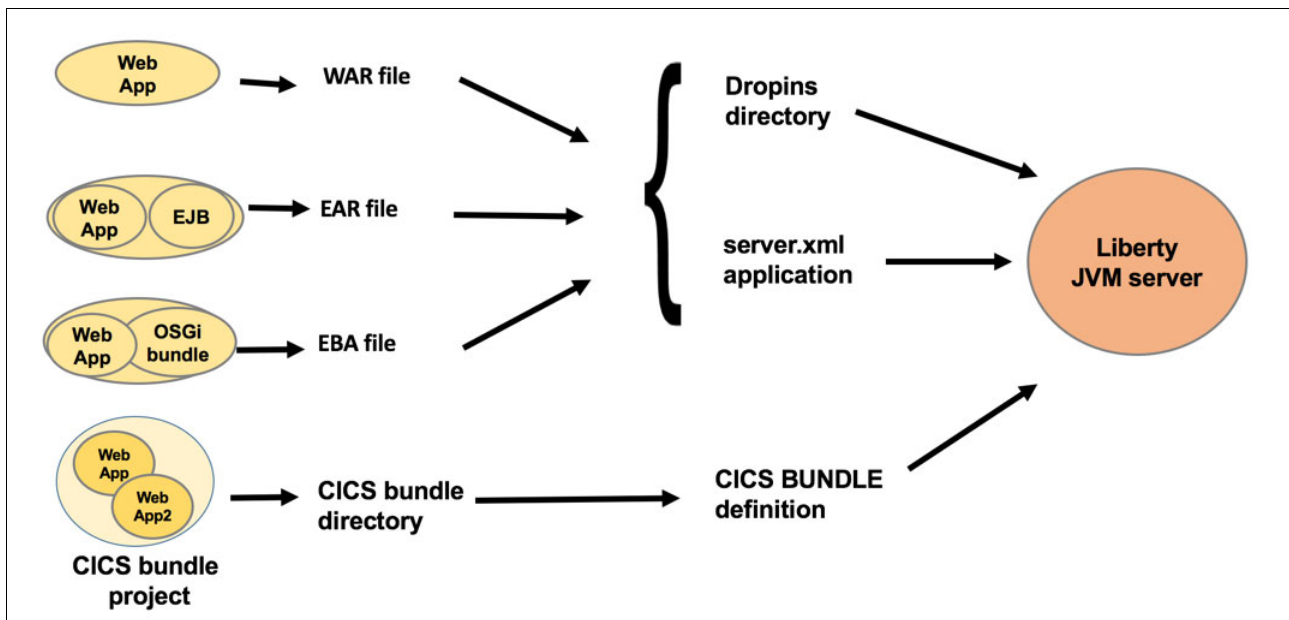


Figure 2-7 CICS Liberty application deployment options

In the following sections, we describe how each of these deployment options can be used.

Common deployment tasks

Whichever one of the three deployment options is used, the following common tasks must be performed before our restapp application can be deployed:

- ▶ A Liberty JVM server must be configured. For more information, see Chapter 1, “Installation and configuration ” on page 1.
- ▶ The JAX-RS feature must be enabled in Liberty. If this feature is not enabled, the JAX-RS annotations in the REST classes are not recognized and the REST URIs are not found, which causes 404 error responses to be returned.
- ▶ Any dependent CICS resources that are used by the application must be available. For restapp, the COBOL program EDUCHAN that is linked to by the reverse service should also be download from the GitHub website and deployed to a library in the CICS region.

2.2.1 Deployment by using Liberty dropins

The simplest options to deploy a web application to Liberty in CICS is to copy the web application archive (.war or .ear file) to the Liberty dropins directory. Liberty then automatically installs and activates the application.

Note: The dropins mechanism offers limited qualities of service. It does not set security roles and it incurs more runtime CPU cost in directory scanning. For these reasons, the dropins deployment option is best-suited to development systems.

This option is available only if the dropins directory deployment option was enabled in the server.xml for Liberty. The CICS auto-generated server.xml includes a default applicationMonitor configuration element with this option disabled, as shown in Example 2-1.

Example 2-1 applicationMonitor in server.xml

```
<applicationMonitor
  dropins="dropins"
  dropinsEnabled="false"
  pollingRate="5s"
  updateTrigger="disabled" />
```

Complete the following steps to deploy the restapp sample by using the dropins method:

1. Enable the dropins capability by setting dropinsEnabled="true" and adding the dropins directory to the applicationMonitor element in server.xml, as shown in Example 2-2.

Example 2-2 Enabling dropins in server.xml

```
<applicationMonitor
  dropins="/var/cicsts/SC8CICS7/ITS0JVM1/dropins"
  dropinsEnabled="true"
  pollingRate="5s"
  updateTrigger="disabled" />
```

Note: At this stage, the attribute updateTrigger="disabled" causes Liberty to not detect application updates in the dropins folder; however, it does allow any new files to be detected and installed.

2. Complete the following steps to export the application as a .war file:
 - a. In CICS Explorer right-click the **com.ibm.cicsdev.restapp** project and select **File** → **Export** → **WAR file**. In the WAR Export window, select **Browse** to choose a suitable local directory for the exported .war file. Then, click **Finish**.
 - b. Transfer the .war file from the workstation to the dropins directory in zFS.
In our example, we used a binary FTP transfer from the export directory to our zFS directory /var/cicsts/SC8CICS7/ITS0JVM1/dropins.
 - c. Validate that the CICS region user ID includes read access to the com.ibm.cicsdev.restapp.war file in the dropins directory.
We modified the file owner to our CICS region user ID (CICSREGN) and the group owner to our CICS admin group (ADMIN), and verified both had read/write permissions by using ISPF option 3.4 (see Figure 2-8).

Menu Utilities View Options Help											
z/OS UNIX Directory List										Row 1 to 3 of 3	
Command ==> █										Scroll ==> PAGE	
Pathname . : /cicsts/SC8CICS7/ITS0JVM1/dropins											
EUID . . . : 190											
Command	Filename	Message	Type	Permission	Audit	Ext	Fmat	Owner	Group	Links	Size Modified

_____	.		Dir	rw-rw-r-x	fff--	----		CICSREGN	CICSADMN	2	8192 2017/11/14 04:11:55
_____	..		Dir	rw-rw-r-x	fff--	----		CICSREGN	CICSADMN	5	8192 2017/11/14 04:24:35
_____	com.ibm.cicsdev		File	rw-rw----	fff--	--s-	----	CICSREGN	CICSADMN	1	15234 2017/11/14 04:02:31
***** Bottom of data *****											
Time zone EST5EDT is used to calculate the displayed date and time values.											
F1=Help F2=Split F3=Exit F4=Expand F7=Backward F8=Forward F9=Swap F10=Actions F12=Cancel											

Figure 2-8 ISPF z/OS UNIX directory list of dropins

3. Check the Liberty messages.log for the application enablement messages. A few seconds after copying the .war file to the dropins directory, messages that are similar to messages that are shown in Example 2-3 should appear in the Liberty messages.log file.

Example 2-3 Liberty messages after enabling dropins

```
[10/27/17 6:58:07:239 EDT] 000067fa com.ibm.ws.webcontainer.osgi.webapp.WebGroup
I SRVE0169I: Loading Web Module: com.ibm.cicsdev.restapp.
[10/27/17 6:58:07:240 EDT] 000067fa com.ibm.ws.webcontainer
I SRVE0250I: Web Module com.ibm.cicsdev.restapp has been bound to default_host.
[10/27/17 6:58:07:240 EDT] 000067fa com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0016I: Web application available (default_host):
http://wtsc80.cpolab.ibm.com:57080/com.ibm.cicsdev.restapp/
[10/27/17 6:58:07:240 EDT] 000067fa com.ibm.ws.app.manager.AppMessageHelper
A CWWKZ0001I: Application com.ibm.cicsdev.restapp started in 0.040 seconds.
```

Now, you can test the restapp cicsinfo service by sending an HTTP GET request to the application by using the following URI in a web browser:

http://wtsc80.cpolab.ibm.com:57080/com.ibm.cicsdev.restapp/rest/cicsinfo

This test should result in a JSON response that includes information about the CICS region, as shown in Figure 2-9 on page 35.

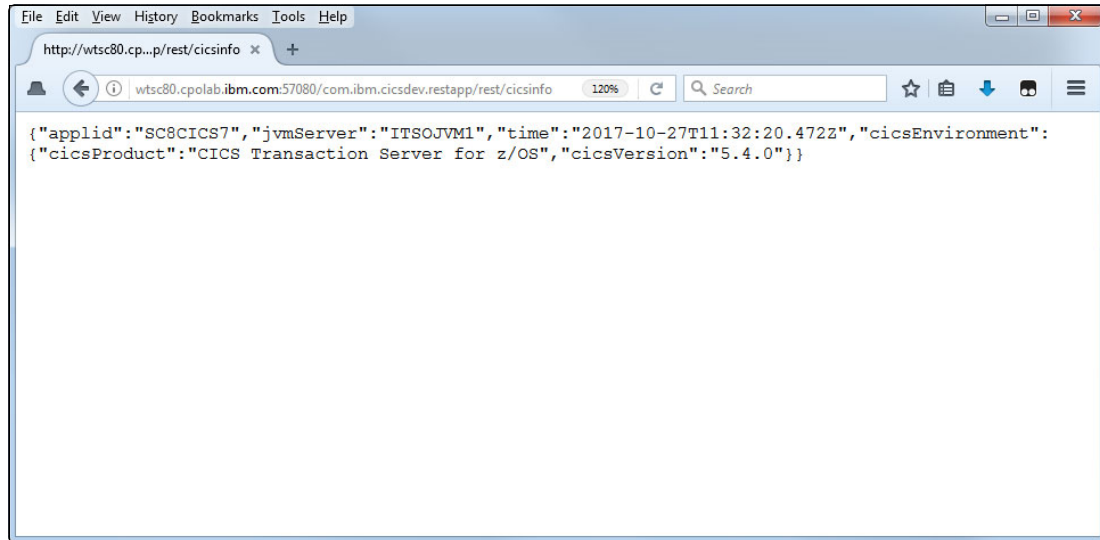


Figure 2-9 Liberty: restapp cicsinfo service

Further scanning of the dropins directory can be performed to detect application updates or the addition and removal of applications to the dropins directory. This feature is enabled by setting `updateTrigger` to `polled`. The `pollingRate` attribute determines how often the dropins directory is scanned for updates (see Example 2-4).

Example 2-4 Enabling `updateTrigger` for dropins

```
<applicationMonitor
  dropins="/var/cicsts/SC8CICS7/ITSOJVM1/dropins"
  dropinsEnabled="true"
  pollingRate="5s"
  updateTrigger="polled" />
```

Removing web applications from the dropins directory

A web application that was deployed by using the dropins mechanism can be removed by deleting the application archive from the dropins directory. When the deletion is detected by Liberty, the application is stopped and the messages that are shown in Example 2-5 are written to the Liberty `messages.log` file.

Example 2-5 Application removal by using dropins

```
[10/27/17 7:21:18:669 EDT] 00006516 com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0017I: Web application removed (default_host):
http://wtsc80.cpolab.ibm.com:57080/com.ibm.cicsdev.restapp/
[10/27/17 7:21:18:682 EDT] 00006516 com.ibm.ws.app.manager.AppMessageHelper
A CWWKZ0009I: The application com.ibm.cicsdev.restapp has stopped successfully.
[10/27/17 7:21:18:822 EDT] 00006550
com.ibm.ws.webcontainer.osgi.mbeans.PluginGenerator          I SRVE9103I: A
configuration file for a web server plugin was automatically generated for this
server at /var/cicsts/SC8CICS7/wlp/servers/itsowlp1/logs/state/plugin-cfg.xml.
```

Subsequent requests to the application fail, and an HTTP GET to the restapp returns the Context Root Not Found error page to the browser (see Figure 2-10 on page 36) because the web application context root `com.ibm.cicsdev.restapp` no longer is known.

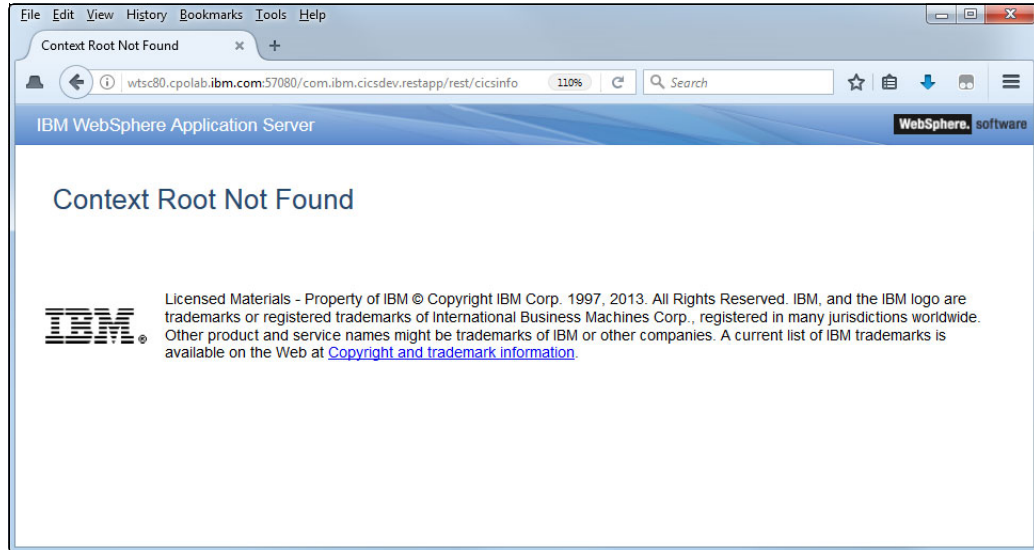


Figure 2-10 Context Root Not Found error message

Updating applications by using dropins directory

An application can be dynamically updated in the dropins directory by overwriting the version of an application with a new version in the directory. To enable this function, the `updateTrigger` on the `applicationMonitor` element must be set to `polled`.

When Liberty detects that an application archive is updated in the dropins directory, it stops the old version and loads the new version of the application. Messages similar to the messages that are shown in Example 2-6 are written to the `messages.log` file.

Example 2-6 Updating dropins

```
[10/27/17 7:45:30:009 EDT] 00006ea7 com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0017I: Web application removed (default_host):
http://wtsc80.cpolab.ibm.com:57080/com.ibm.cicsdev.restapp/
[10/27/17 7:45:30:023 EDT] 00006ea7 com.ibm.ws.app.manager.AppMessageHelper
A CWWKZ0009I: The application com.ibm.cicsdev.restapp has stopped successfully.
[10/27/17 7:45:30:044 EDT] 000065bf
com.ibm.ws.webcontainer.osgi.mbeans.PluginGenerator          I SRVE9103I: A
configuration file for a web server plugin was automatically generated for this
server at /var/cicsts/SC8CICS7/wlp/servers/itsowlp1/logs/state/plugin-cfg.xml.
[10/27/17 7:45:30:045 EDT] 00006ea7 com.ibm.ws.app.manager.AppMessageHelper
I CWWKZ0018I: Starting application com.ibm.cicsdev.restapp.
[10/27/17 7:45:30:092 EDT] 00006ea7 com.ibm.ws.webcontainer.osgi.webapp.WebGroup
I SRVE0169I: Loading Web Module: com.ibm.cicsdev.restapp.
[10/27/17 7:45:30:092 EDT] 00006ea7 com.ibm.ws.webcontainer
I SRVE0250I: Web Module com.ibm.cicsdev.restapp has been bound to default_host.
[10/27/17 7:45:30:092 EDT] 00006ea7 com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0016I: Web application available (default_host):
http://wtsc80.cpolab.ibm.com:57080/com.ibm.cicsdev.restapp/
[10/27/17 7:45:30:092 EDT] 00006ea7 com.ibm.ws.app.manager.AppMessageHelper
A CWWKZ0003I: The application com.ibm.cicsdev.restapp updated in 0.047 seconds.
```

2.2.2 Deployment as an application element in server.xml

The second deployment option we describe uses an application element in the Liberty `server.xml` file. This option is highly customizable and allows the context root, security roles, and classloader definitions to be modified. For more information, see 2.3.1, “Shared libraries” on page 53 and 2.3.2, “Global libraries” on page 54.

Liberty supports several different elements that can be used in `server.xml` to define applications. You can use the `<application>` element, or you can use one of specific elements `<webApplication>`, `<osgiApplication>`, or `<ejbApplication>` as appropriate to the file archive type.

We used the `<webApplication>` element that is shown in Example 2-7 in our `server.xml` to install `restapp` into our Liberty JVM server. The application bindings that are shown in the `<application-bnd>` element use the same pattern as used in the `installedApps.xml`, which uses the `cicsAllAuthenticated` role to grants access to all authenticated Liberty users. However, these permissions can be further customized as described in Chapter 7, “Securing web applications” on page 175.

Example 2-7 Liberty server.xml - webApplication element

```
<webApplication id="com.ibm.cicsdev.restapp"
  location="/var/cicsts/SC8CICS7/ITS0JVM1/apps/com.ibm.cicsdev.restapp.war"
  name="com.ibm.cicsdev.restapp">
  <application-bnd>
    <security-role name="cicsAllAuthenticated">
      <special-subject type="ALL_AUTHENTICATED_USERS" />
    </security-role>
  </application-bnd>
</webApplication>
```

Using a server.xml include file

A variation of this deployment method is to define the application in a separate XML fragment file, and then include this fragment in `server.xml` by using the `<include>` element in the `server.xml`. In our configuration, we added the following include into our `server.xml` file:

```
<include location="${server.output.dir}/restapp.xml"/>
```

Although the referenced `restapp.xml` fragment needs to contain only the `webApplication` element as shown in Example 2-7, it can also contain more resources, such as library definitions or features.

For more information about how we used `server.xml` include files, see Chapter 5, “Connecting to IBM MQ by using JMS” on page 113.

Configuration dropins directory

An alternative to using a specific include file is to use the Liberty server configuration dropins directory. This directory is a specifically named directory structure, which is automatically scanned by Liberty to dynamically add `server.xml` configuration fragments. This technique can allow self-contained `server.xml` fragments to be deployed for an application without needing to modify the configuration files.

To enable this function, we created the configDropins directory in the /var/cicsts/SC8CICS2/wlp/servers/itsowlp1/ directory. Then, we created the overrides directory within this directory, as shown in Figure 2-11.

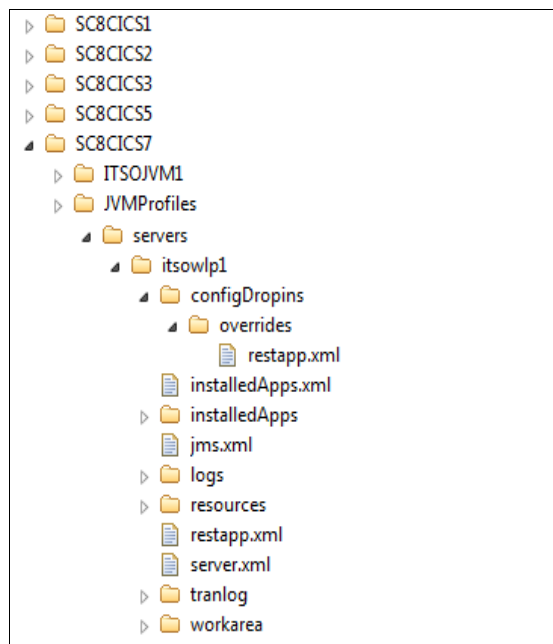


Figure 2-11 configDropins directory structure

We then created the restapp.xml fragment that includes a <server/> element and a <webApplication/> element, as shown in Example 2-8.

Example 2-8 configDropins/overrides/restapp.xml fragment

```
<server>
  <webApplication id="com.ibm.cicsdev.restapp"
    location="/var/cicsts/SC8CICS7/ITS0JVM1/apps/com.ibm.cicsdev.restapp.war"
    name="com.ibm.cicsdev.restapp">
    <application-bnd>
      <security-role name="cicsAllAuthenticated">
        <special-subject type="ALL_AUTHENTICATED_USERS"/>
      </security-role>
    </application-bnd>
  </webApplication>
</server>
```

When the `restapp.xml` file was deployed to the `configDropins` directory, we noted that the Liberty server scanned the `configDropins/overrides` directory and installed the new `restapp` application, as shown in the `messages.log` file (see Example 2-9).

Example 2-9 Liberty messages with `configDropins` enabled

```
[11/14/17 10:02:55:983 EST] 00000068 com.ibm.ws.config.xml.internal.ConfigRefresher
A CWWKG0016I: Starting server configuration update.
[11/14/17 10:02:55:988 EST] 00000068 com.ibm.ws.config.xml.internal.XMLConfigParser
A CWWKG0028A: Processing included configuration resource:
/var/cicsts/SC8CICS2/wlp/servers/itsowlp1/installedApps.xml
[11/14/17 10:02:55:992 EST] 00000068 com.ibm.ws.config.xml.internal.ServerXMLConfiguration
A CWWKG0093A: Processing configuration drop-ins resource:
/var/cicsts/SC8CICS2/wlp/servers/itsowlp1/configDropins/overrides/restapp.xml
[11/14/17 10:02:56:019 EST] 00000068 com.ibm.ws.config.xml.internal.ConfigRefresher
A CWWKG0017I: The server configuration was successfully updated in 0.035 seconds.
[11/14/17 10:02:56:034 EST] 00000055 com.ibm.ws.app.manager.AppMessageHelper
I CWWKZ0018I: Starting application com.ibm.cicsdev.restapp.
[11/14/17 10:02:56:118 EST] 00000055 com.ibm.ws.webcontainer.osgi.webapp.WebGroup
I SRVE0169I: Loading Web Module: com.ibm.cicsdev.restapp.
[11/14/17 10:02:56:118 EST] 00000055 com.ibm.ws.webcontainer
I SRVE0250I: Web Module com.ibm.cicsdev.restapp has been bound to default_host.
[11/14/17 10:02:56:119 EST] 00000055 com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0016I: Web application available (default_host):
http://wtsc80.cpolab.ibm.com:52080/com.ibm.cicsdev.restapp/
[11/14/17 10:02:56:119 EST] 00000055 com.ibm.ws.app.manager.AppMessageHelper
A CWWKZ0001I: Application com.ibm.cicsdev.restapp started in 0.085 seconds.
```

Removing a Liberty application

A web application that is deployed by using the Liberty application definition mechanism is removed by deleting the application element from `server.xml` or any included fragment. If the `include` method was used, it is sufficient to delete the `include` reference in the `server.xml`.

Updating a Liberty application

An application can be dynamically updated in Liberty by overwriting the deployed archive file with a new version. To enable this function, the `updateTrigger` attribute on the `<applicationMonitor>` element must be set to `polled` as shown in the following example:

```
<applicationMonitor pollingRate="5s" updateTrigger="polled" />
```

The use of this technique causes the application to be stopped and then started when the polling mechanism detects that the `.war` file was updated.

Note: Updating Liberty applications requires a stop and restart of the web application. During this period, a Context Root Not Found response is returned because the HTTP listener is still active, but the web context root is unavailable.

Deploying with a modified context root

In Liberty, only one application per context root can be deployed. A second version of an application that shares the context root as another installed application fails to install. However, an alternative way to deploy an application update is by deploying a new version of the application to a new location on the file system with a different web context root. This process installs the new version of the application alongside the previous version and allows any updates to be tested by using a different URI for each version.

To demonstrate this method, we installed a new version of restapp into our Liberty server by using the context-root tag on the webApplication element, as shown in Example 2-10. The second version of the restapp was deployed as the com.ibm.cicsdev.restapp2.war and accessed by using the URL `http://wtsc80.cpolab.ibm.com:57080/com.ibm.cicsdev.restapp2/rest/cicsinfo`

Example 2-10 Liberty server.xml: webApplication element

```
<webApplication id="com.ibm.cicsdev.restapp2"
  context-root="com.ibm.cicsdev.restapp2"
  location="/var/cicsts/SC8CICS7/ITS0JVM1/apps/com.ibm.cicsdev.
restapp2.war"
  name="com.ibm.cicsdev.restapp2">
  <application-bnd>
    <security-role name="cicsAllAuthenticated">
      <special-subject type="ALL_AUTHENTICATED_USERS" />
    </security-role>
  </application-bnd>
</webApplication>
```

Other methods can be used to control the web application's context root (such as in the `ibm-web-ext.xml` or `application.xml` deployment descriptors), depending on the type of web application. For more information about these options, see the topic [“Deploying a web application to Liberty”](#) at IBM Knowledge Center.

2.2.3 Deployment in a CICS bundle

The third deployment method you can use for application deployment to Liberty is a CICS bundle project. A CICS bundle project is a specific type of Eclipse project that is supported by the CICS Explorer SDK. It allows a collection of CICS resources (known as *bundle parts*) to be defined within it.

The bundle parts can include one or more Liberty Web applications that are packaged as a .war, .ear, or .eba file archive types. Bundle projects also support various other CICS resources to be defined, such as transactions, programs, files, and URI map definitions.

CICS bundles are defined as BUNDLE resource definitions in the CSD, which then refer to a zFS directory that includes all of the assembled bundle components.

In the following sections, we describe how to define a web application in a CICS bundle, export the bundle directory structure, and install it into CICS. To show some of the advantages of this deployment approach, we create a single CICS bundle to deploy the restapp sample web application and an associated CICS transaction definition and URI map.

The starting point for CICS bundle deployment assumes that the sample web application restapp was downloaded from GitHub and imported into the CICS Explorer development environment, as described in 2.1, “Building the restapp sample ” on page 28. It is also assumed that the WAR was removed from the dropins directory if this method was used as described in 2.2.1, “Deployment by using Liberty dropins” on page 33.

Creating a CICS bundle project

Complete the following steps to create the CICS bundle project:

1. In CICS Explorer, switch to the Java EE perspective and click **New** → **Other**. The Select a wizard window opens, as shown in Figure 2-12.

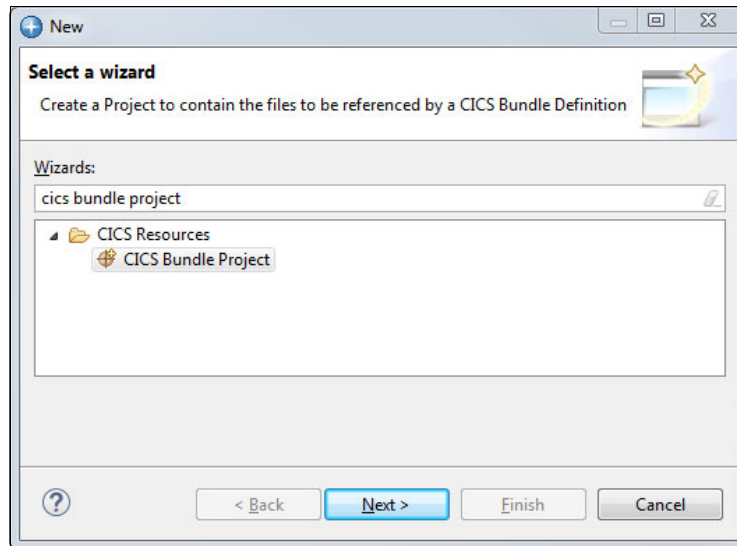


Figure 2-12 Select a wizard: CICS Bundle Project

2. Enter CICS Bundle Project and select **CICS Bundle Project** from the list. Click **Next**.
3. Enter a name for the new project. In our example, we used `com.ibm.cicsdev.restapp.cicsbundle`, as shown in Figure 2-13.

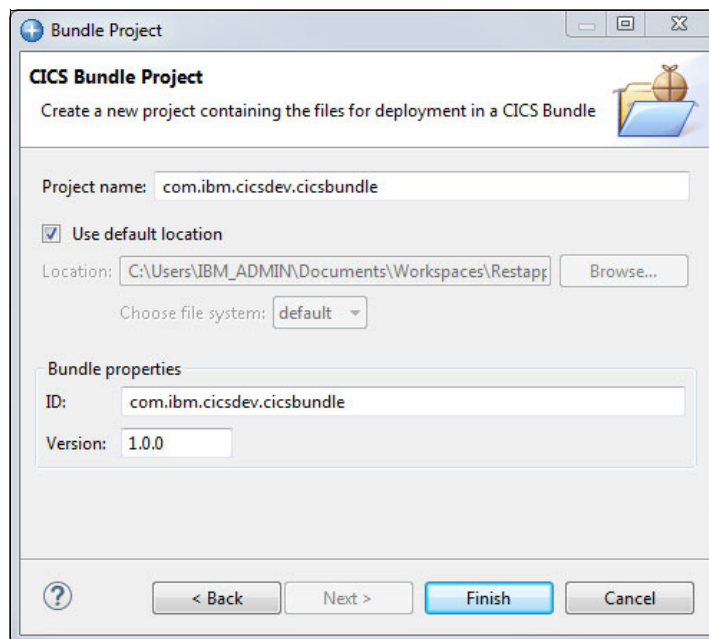


Figure 2-13 New CICS bundle project

4. Click **Finish** to complete the process. The project is created and the bundle manifest editor is displayed, as shown in Figure 2-14 on page 42.

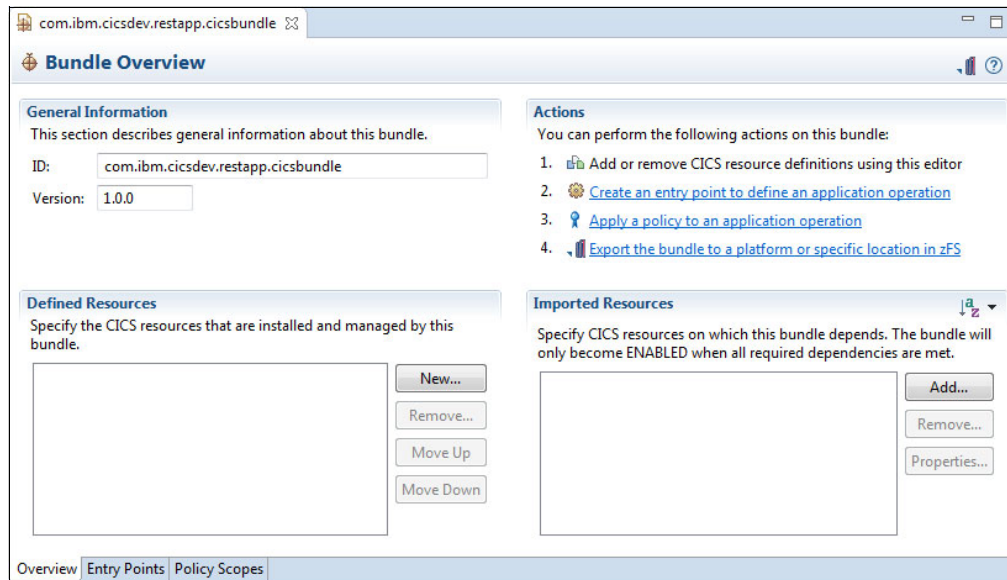


Figure 2-14 Bundle manifest editor for `com.ibm.cicsdev.restapp.bundle`

Adding the web applications to the CICS bundle

The next task is to add the web project to a CICS bundle project, which is required for deployment of the application in a CICS bundle. Complete the following steps:

1. In the CICS bundle editor, the Bundle Overview window includes is a section for Defined Resources. These CICS resources are included in the bundle.
2. Click **New** and select **Dynamic Web Project Include**. The Dynamic Web Project Include window opens and displays the `com.ibm.cicsdev.restapp` project as a potential candidate, as shown in Figure 2-15.

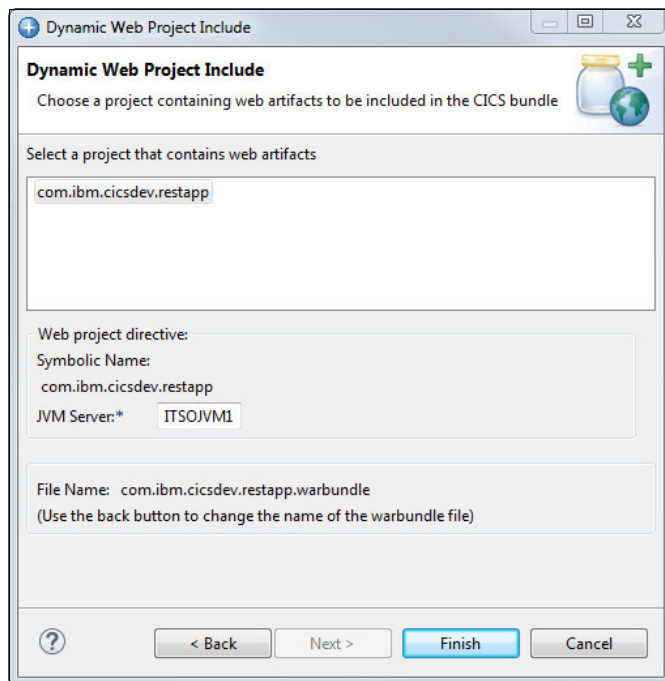


Figure 2-15 Dynamic Web Project Include window

3. Select the project **com.ibm.cicsdev.restapp** and enter the name of the JVM server to use (see Figure 2-16). In our example, we used ITSOJVM1. Click **Finish**.

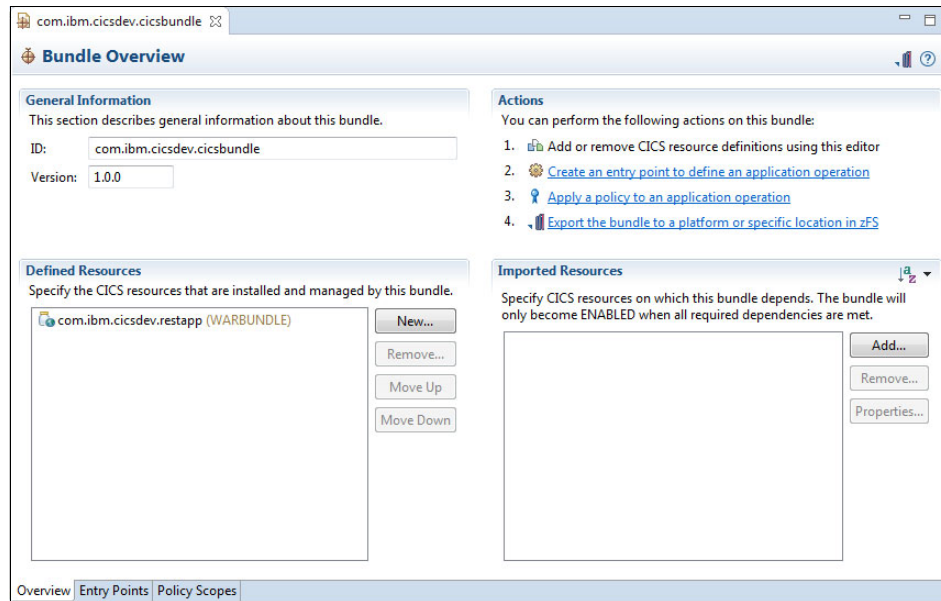


Figure 2-16 Bundle Overview that shows the defined WARBUNDLE part

Adding CICS resources to the CICS bundle

The restapp application requires minimal extra CICS resources for operation. We chose to define a URI map and a specific request processor transaction rather than allow requests to run under the default transaction ID CJSA. We can add these resource definitions to the CICS bundle rather than define them in the CSD, which allows all the application component to be defined in a single package for deployment.

Complete the following steps:

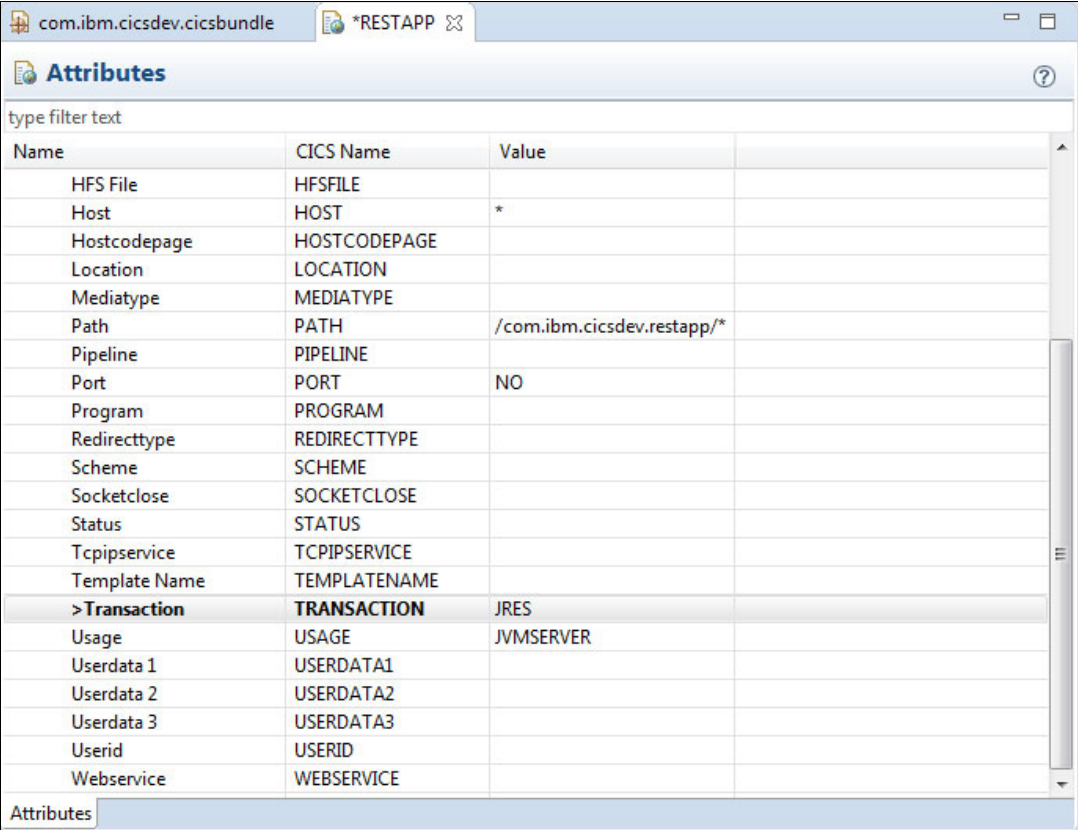
1. In the Defined Resources section of the CICS Bundle Editor, click **New**. Select **URIMAP Definition**. A Create URI Map Definition window opens. Add the following attributes as shown in Figure 2-17 on page 44:
 - Name: RESTAPP
 - Host: *
 - Path: /com.ibm.cicsdev.restapp/*
 - Usage: JVM Server
 - Port: NO

The screenshot shows a 'New URI Map Definition' dialog box. The title bar says 'New URI Map Definition'. The main title is 'Create URI Map Definition' with a subtitle 'Create a URI map definition in a CICS bundle.' The 'Bundle' field is set to 'com.ibm.cicsdev.cicsbundle'. The 'Name' field is 'RESTAPP'. The 'Description' field is 'restapp URI map'. The 'Host' field is '*'. The 'Path' field is '/com.ibm.cicsdev.restapp/*'. Under the 'Usage' section, 'JVM Server' is selected with a radio button. The 'Port' field for 'JVM Server' is 'NO'. There are also options for 'Server (dynamic)', 'Server (static)', 'Client', 'Atom', and 'Pipeline'. At the bottom, there is a checkbox 'Create an application entry point' and a checkbox 'Open editor' which is checked. The 'Finish' and 'Cancel' buttons are at the bottom right.

Figure 2-17 Create URI Map Definition window

2. Click **Finish**. The attribute editor is displayed and shows the resulting URIMAP.

Next, we must modify the transaction ID to which this URI map refers. Click **Transaction** and enter JRES as the value, as shown in Figure 2-18.



The screenshot shows the 'Attributes' dialog box with the following table of attributes:

Name	CICS Name	Value
HFS File	HFSFILE	
Host	HOST	*
Hostcodepage	HOSTCODEPAGE	
Location	LOCATION	
Mediatype	MEDIATYPE	
Path	PATH	/com.ibm.cicsdev.restapp/*
Pipeline	PIPELINE	
Port	PORT	NO
Program	PROGRAM	
Redirecttype	REDIRECTTYPE	
Scheme	SCHEME	
Socketclose	SOCKETCLOSE	
Status	STATUS	
Tcpipservice	TCPIPSERVICE	
Template Name	TEMPLATENAME	
> Transaction	TRANSACTION	JRES
Usage	USAGE	JVMSEVER
Userdata 1	USERDATA1	
Userdata 2	USERDATA2	
Userdata 3	USERDATA3	
Userid	USERID	
Webservice	WEBSERVICE	

Figure 2-18 Editing the URI map attributes

3. Close the attribute editor and click **Yes** to save the changes when prompted.

Next, we must create a TRANSACTION definition for the JRES transaction that we referred to in our URI map definition.

4. In the Defined Resources section of the CICS Bundle Editor, click **New** then, select **Transaction Definition**. The Create Transaction Definition window opens. Add the following attributes that are shown in Figure 2-19. You must enter the program name as DFHSJTHP because it handles the security checking of inbound Java EE requests to the Liberty server.

New Transaction Definition

Create a transaction definition in a CICS bundle.

Bundle: com.ibm.cicsdev.cicsbundle

Name:* JRES

Description: Transaction ID for restapp Java service

Program Name: DFHSJTHP

Remote System Name: Remote Transaction Name:

☐ Create an application entry point

Operation name:

☒ Open editor

Finish Cancel

Figure 2-19 Create Transaction Definition window

Exporting the CICS bundle to zFS

We must build the bundle and export it to z/OS. We can build the bundle from CICS Explorer or use the CICS build toolkit to automate this process as a batch job. Both methods automatically build the Java web application in a .war file and include this file with the transaction and URI map in the assembled CICS bundle directory on zFS.

In CICS Explorer, complete the following steps:

1. Ensure that a connection to z/OS is configured. CICS Explorer can use FTP, RSE, or z/OS MF to export CICS bundles. Complete the following steps to create an FTP connection:
 - a. In the CICS SM perspective, open the Host Connections view. Click **Add** → **z/OS FTP** to define a new connection (see Figure 2-20 on page 47).

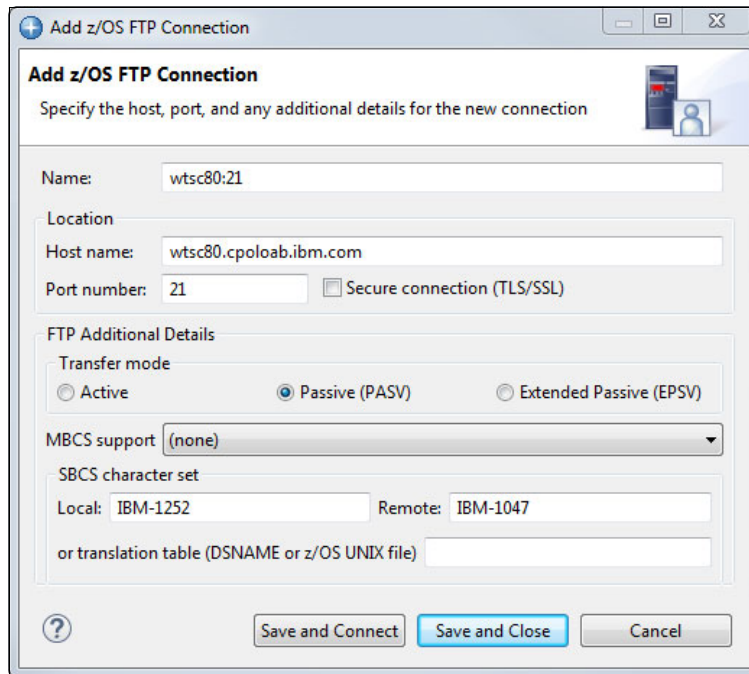


Figure 2-20 Add z/OS FTP Connection

- b. Enter the host name of the FTP server and add the default remote EBCDIC character set. In our example, we supplied the value IBM-1047 for US EBCDIC.

Tip: Instead of setting the default EBCDIC encoding for each connection in CICS Explorer, this encoding can be set on z/OS by using the SBDATACONN parameter in the SYS1.TCPPARMS(FTPDATA) member, as shown in the following example:

```
SBDATACONN (IBM-1047,IS08859-1)
```

- c. Click **Save and Connect**. You are prompted to create a user ID and the connection status icon should now be green.
2. Now a connection is available to which we can export the CICS bundle project to zFS. Complete the following steps:
 - a. Using the Project Explorer in the Java EE perspective, right-click the CICS bundle project **com.ibm.cicsdev.restapp.bundle** and select **Export Bundle Project to z/OS UNIX File System**.
 - b. Select **Export to a specific location in the file system**. Click **Next**.
 - c. The Export to z/OS UNIX File System window opens. For Parent Directory, enter the target zFS directory you chose to contain your exported CICS bundles (see Figure 2-21 on page 48 and Figure 2-22 on page 48).
 - d. Click **Finish** to complete the export.

For our example, we used the parent directory `/var/cicsts/SC8CICS7/ITS0JVM1/bundles/` and CICS Explorer then, we created the sub-directory `com.ibm.cicsdev.restapp.cicsbundle_1.0.0` within this directory that contains the following components (see Figure 2-21 on page 48):

- `com.ibm.cicsdev.restapp.war`: WAR archive
- `com.ibm.cicsdev.restapp.warbundle`: XML WAR bundle part descriptor
- `JRES.transaction`: XML transaction bundle part descriptor

- RESTAPP.urimap: XML URI map bundle part descriptor
- META-INF/cics.xml: XML CICS bundle manifest

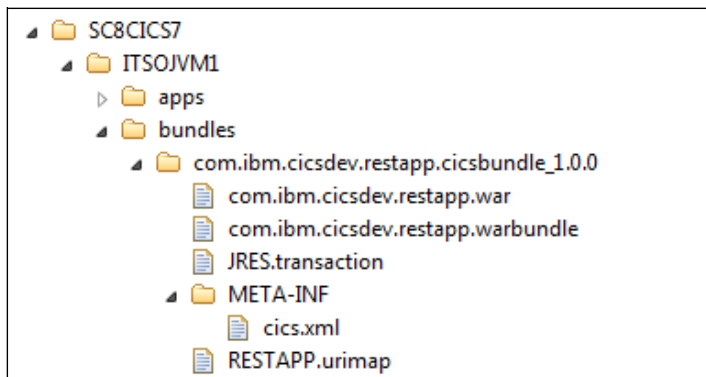


Figure 2-21 Restapp bundle directory structure

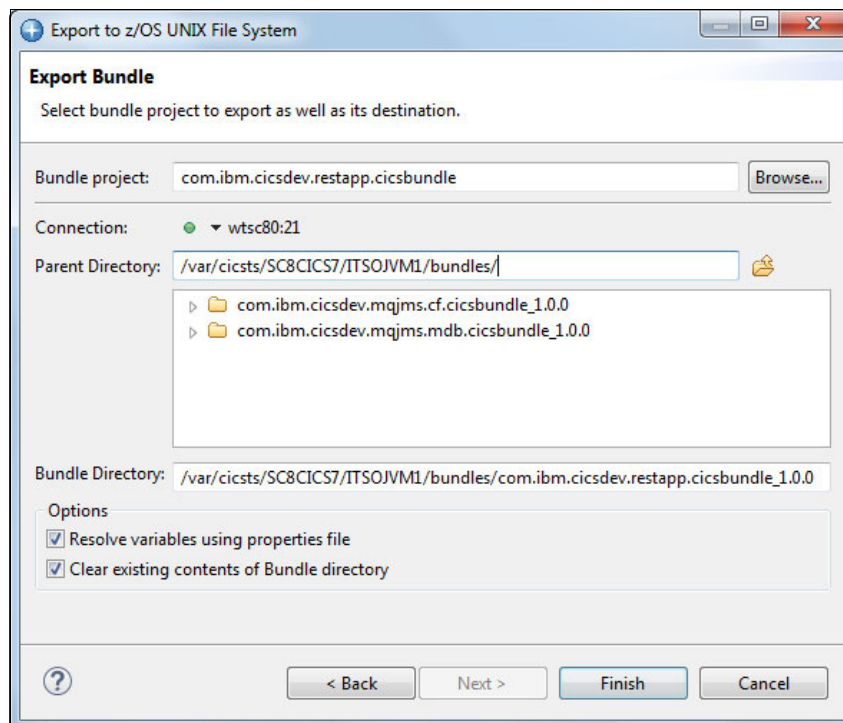


Figure 2-22 Export to z/OS UNIX File System

Tip: When bundles are exported to zFS, you can copy the fully qualified target bundle directory name to the clipboard before completing the export process. This copying is useful when the CICS BUNDLE resource is defined in the next step.

Installing the CICS bundle

Complete the following steps to install the CICS bundle:

1. A CICS BUNDLE resource definition must be defined and installed to install the exported CICS bundle project into our target CICS region. You can use various tools to perform this task, including CEDA, CICS Explorer, DFHCSDUP, or DFHDPLOY. We used CEDA, as shown in Figure 2-23 to add the BUNDLE definition to our CSD.

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA ALTER Bundle( RESTAPP )
  Bundle      : RESTAPP
  Group       : RESTAPP
  DESCRIPTION ==> RESTAPP WEB APPLICATION BUNDLE
  Status      ==> Enabled          Enabled | Disabled
  BUNDLEDIR   ==> /var/cicsts/SC8CICS7/ITS0JVM1/bundles/com.ibm.cicsdev.rest
  (Mixed Case) ==> app.cicsbundle_1.0.0
  ==>
  ==>
  BASESCOPE   ==>
  (Mixed Case) ==>
  ==>
  ==>
  ==>
  DEFINITION SIGNATURE
  Definetime  : 10/31/17 10:03:39
+ CHANGETime  : 10/31/17 10:03:39
                                                    SYSID=SC87 APPLID=SC8CICS7

PF 1 HELP 2 COM 3 END                6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 2-23 CEDA - DEFINE BUNDLE

The CICS BUNDLE features an attribute that is named BUNDLEDIR, which is the fully qualified path to the bundle directory in zFS where the bundle was exported and should be set to the bundle root directory, as shown in the following example:

```
/var/cicsts/SC8CICS7/ITS0JVM1/bundles/com.ibm.cicsdev.restapp.cicsbundle_1.0.0
```

2. The BUNDLE definition must be installed and added to a CSD group list to ensure it is also installed on a CICS cold start. We installed the BUNDLE resource and checked the Liberty messages.log file for the messages SRVE0250I and CWWKT0016I, which indicate that com.ibm.cicsdev.restapp is now available, as shown in Example 2-11.

Example 2-11 Liberty messages.log - installing CICS bundle

```
[10/31/17 10:51:37:448 EDT] 0002a80c com.ibm.ws.webcontainer
I SRVE0250I: Web Module com.ibm.cicsdev.restapp has been bound to default_host.
[10/31/17 10:51:37:448 EDT] 0002a80c com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0016I: Web application available (default_host):
http://wtsc80.cpolab.ibm.com:57080/com.ibm.cicsdev.restapp/
```

- In CICS Explorer, we can now validate that the URI map and transaction resources were installed. Switch to the CICS SM perspective in CICS Explorer and select **Operations** → **Bundles**. The RESTAPP bundle is shown. It should be in the ENABLED status with the Enabled Count, Part Count, and Target Count set to 3 because three bundle parts are used (see Figure 2-24).

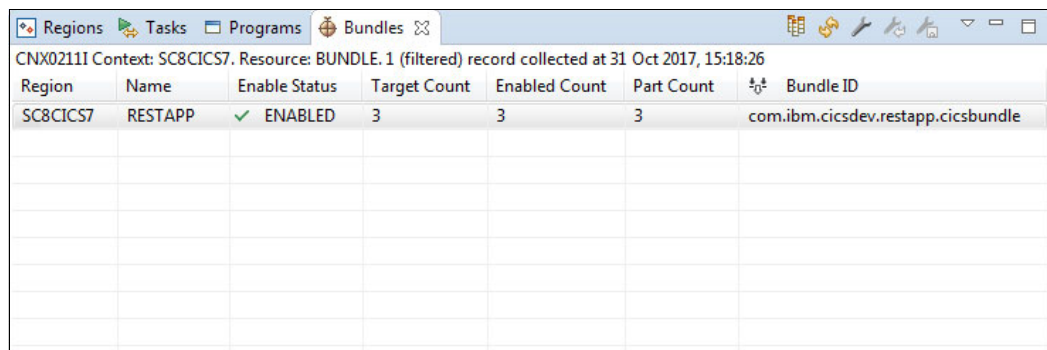


Figure 2-24 shows the CICS Explorer interface with the 'Bundles' tab selected. The title bar indicates 'CNX0211I Context: SC8CICS7. Resource: BUNDLE. 1 (filtered) record collected at 31 Oct 2017, 15:18:26'. The table below displays the details of the RESTAPP bundle.

Region	Name	Enable Status	Target Count	Enabled Count	Part Count	Bundle ID
SC8CICS7	RESTAPP	✓ ENABLED	3	3	3	com.ibm.cicsdev.restapp.cicsbundle

Figure 2-24 RESTAPP bundle

To validate the bundle parts, right-click the **RESTAPP** bundle and select **Show Bundle Parts**. The components that are in the bundle are shown (see Figure 2-25). In our example, the WAR bundlepart, JRES transaction, and RESTAPP URI map are all marked as ENABLED.

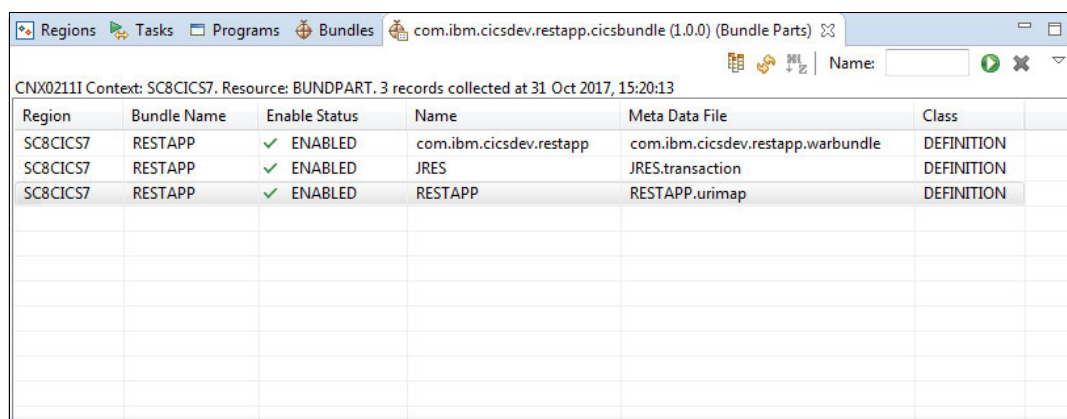


Figure 2-25 shows the CICS Explorer interface with the 'Bundle Parts' tab selected for the RESTAPP bundle. The title bar indicates 'com.ibm.cicsdev.restapp.cicsbundle (1.0.0) (Bundle Parts)'. The table below displays the details of the bundle parts.

Region	Bundle Name	Enable Status	Name	Meta Data File	Class
SC8CICS7	RESTAPP	✓ ENABLED	com.ibm.cicsdev.restapp	com.ibm.cicsdev.restapp.warbundle	DEFINITION
SC8CICS7	RESTAPP	✓ ENABLED	JRES	JRES.transaction	DEFINITION
SC8CICS7	RESTAPP	✓ ENABLED	RESTAPP	RESTAPP.urimap	DEFINITION

Figure 2-25 Show bundle parts

- We can now test the restapp again by sending a simple HTTP request to the application by using the following URI:

`http://wtsc80.cpolab.ibm.com:57080/com.ibm.cicsdev.restapp/rest/cicsinfo`

This test returns a JSON response that includes information about the CICS region, as shown in Figure 2-26 on page 51.

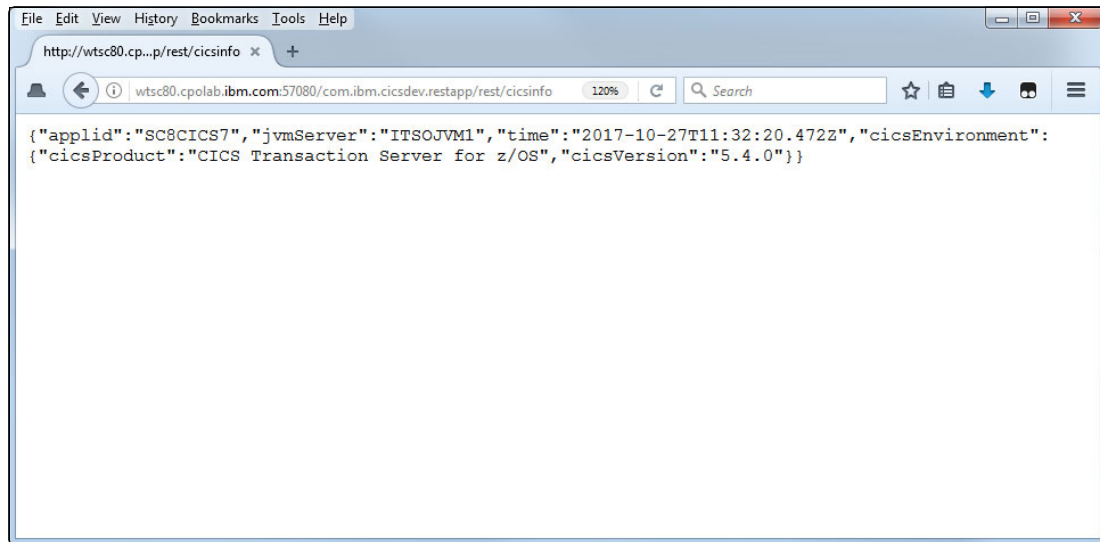


Figure 2-26 Liberty - restapp - cicsinfo service

Note: The installation mechanism that CICS uses for web applications in a CICS bundle involves copying the web application archive into the `/installedApps` directory and then adding an application element for each Liberty application to the `installedApps.xml`, which is included as part of the `server.xml`. The addition of new application elements in the `installedApps.xml` causes Liberty to install the web applications by using its standard application installation process.

Removing a CICS bundle

To uninstall a web application that is installed by using a CICS bundle, the CICS bundle must be disabled. This process disables all of the resources that are defined by the bundle, including web application bundle parts, which stop the applications within Liberty.

Disabling a web application causes CICS to remove the `.war` file from the `/installedApps` directory and to remove the application entry from the `installedApps.xml` file. This process in turn causes Liberty to remove the web application because the `installedApps.xml` is included as part of the `server.xml` configuration.

Updating a CICS bundle

To update a web application that was deployed as a CICS bundle, update the application in the development environment. Then, reexport the CICS bundle project to zFS by following the deployment instructions that are described in 2.2.1, “Deployment by using Liberty dropsins” on page 33.

Then, disable and re-enable the BUNDLE resource by using the CICS SPI. This task can be completed by using one of the following methods:

- ▶ **CEMT SET BUNDLE() DISABLED** and **ENABLED** commands
- ▶ CICS Explorer Bundle Operations view
- ▶ Batch job by using **DFHDPLOY SET BUNDLE STATE(DISABLED)** and **(ENABLED)** commands
- ▶ CICS SPI program by using **EXEC CICS SET BUNDLE() DISABLED** and **ENABLED**

These commands cause the web application to be removed from the `installedApps.xml` file when the bundle is disabled, and then, added again when the bundle is enabled. Liberty messages that indicate that this process occurred are written to the `Liberty messages.log`, as shown in Example 2-12.

Example 2-12 Liberty messages.log - updating a CICS bundle

```
[11/2/17 11:46:23:345 EDT] 00003282 com.ibm.ws.config.xml.internal.ConfigRefresher
A CWWKG0016I: Starting server configuration update.
[11/2/17 11:46:23:348 EDT] 00003282 com.ibm.ws.config.xml.internal.XMLConfigParser
A CWWKG0028A: Processing included configuration resource:
/var/cicsts/SC8CICS7/wlp/servers/itsowlp1/installedApps.xml
[11/2/17 11:46:23:374 EDT] 00003282 com.ibm.ws.config.xml.internal.ConfigRefresher
A CWWKG0017I: The server configuration was successfully updated in 0.029 seconds.
[11/2/17 11:46:23:390 EDT] 000027b3 com.ibm.ws.app.manager.AppMessageHelper
I CWWKZ0018I: Starting application com.ibm.cicsdev.restapp.
[11/2/17 11:46:23:440 EDT] 000027b3 com.ibm.ws.webcontainer.osgi.webapp.WebGroup
I SRVE0169I: Loading Web Module: com.ibm.cicsdev.restapp.
[11/2/17 11:46:23:440 EDT] 000027b3 com.ibm.ws.webcontainer
I SRVE0250I: Web Module com.ibm.cicsdev.restapp has been bound to default_host.
[11/2/17 11:46:23:440 EDT] 000027b3 com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0016I: Web application available (default_host):
http://wtsc80.cpolab.ibm.com:57080/com.ibm.cicsdev.restapp/
```

Warning: If the CICS bundle contains any other bundle parts aside from a Liberty web application, these bundle parts are updated only if the CICS bundle is discarded and reinstalled.

2.2.4 Comparison of the deployment options

The three deployment options we described can all achieve the basic effect of deploying and activating a web application in a Liberty JVM server. However, each option features different characteristics in other respects.

Liberty dropins directory

This option includes the following features:

- ▶ Requires minimal setup
- ▶ Not integrated with CICS security because of a lack of security role support
- ▶ Requires dropins scanning mechanism to be enabled
- ▶ Does not support customization of the application element
- ▶ Best-suited to development systems
- ▶ Deployment artifact is a single Java archive

Liberty application element in server.xml

This option includes the following features:

- ▶ Highly customizable
- ▶ Supports custom class loaders, security roles, and context root modification
- ▶ Deployment artifact is a single Java archive

CICS bundles

This option includes the following features:

- ▶ Integrated with CICS Explorer, CICS build toolkit, and DFHDPLOY tooling
- ▶ Supports packaging of multiple applications and CICS resources in the same bundle
- ▶ CICS SET BUNDLE SPI and CICS Explorer can be used to manage lifecycle
- ▶ Does not support customization of web application elements
- ▶ Deployment artifact is a directory structure

2.3 Advanced deployment options

In this section, we describe some of the other options that are available to customize the deployment process for web applications in a Liberty JVM server.

2.3.1 Shared libraries

Shared libraries are files that are used by multiple applications. You can use shared libraries and global libraries to reduce the number of duplicate library files on your system.

Shared libraries are named by using a `<library>` element and then specified on an individual application by modifying the application class loader. This configuration enables libraries to be shared between multiple applications, or other libraries to be added to the application.

As shown in Example 2-13, the `userlib.jar` library was added to our `restapp` by using the `<classloader>` child element on the `<webApplication>`.

Example 2-13 Adding library to class loader

```
<webApplication id="com.ibm.cicsdev.restapp"
  location="/var/cicsts/SC8CICS7/ITS0JVM1/apps/com.ibm.cicsdev.restapp.war"
  name="com.ibm.cicsdev.restapp">
  <classloader commonLibraryRef="privatelib" />
</webApplication>
<library id="privatelib">
  <fileset dir="/var/cicsts/SC8CICS7/ITS0JVM1/apps"
    includes="userlib.jar" />
</library>
```

Note: OSGi applications do not use shared libraries. Instead, they should use a shared bundle repository for sharing common libraries between applications.

2.3.2 Global libraries

When CICS bundle-defined applications are used, the application elements that are automatically created by CICS cannot be modified. However, you can use a global library to make a library available to all deployed applications, including web applications that are defined in CICS bundles.

As shown in Example 2-14, the library is defined by using `id=global`, which indicates that it is a global library that can be used by any web application that is deployed into the Liberty server.

Example 2-14 Liberty server.xml - global library

```
<library id="global">
  <fileset dir="/var/cicsts/SC8CICS7/ITS0JVM1/apps"
    includes="library.jar" />
</library>
```

For more information about how we used a global library in a web application to make the DB2 libraries available to our SQLJ application, see Chapter 4, “Connecting to Db2 by using JDBC” on page 91.

Note: A class in a global library takes precedence over the same class that is deployed within an application. Therefore, it must be used with caution for any application that might require modification.

2.3.3 Deploying a prebuilt Java archive in a CICS bundle

If you want to deploy a web application in a CICS bundle by using a Java archive that is built or is being ported from another application server, add the archive directly into a CICS bundle project. Complete the following steps:

1. In CICS Explorer, switch to the Java EE perspective and click **File** → **New** → **Other**. The Select a wizard window opens.
2. Enter **CICS Bundle Project** in the search field and select **CICS Bundle Project** from the resulting list, as shown in Figure 2-27 on page 55. Click **Next**.

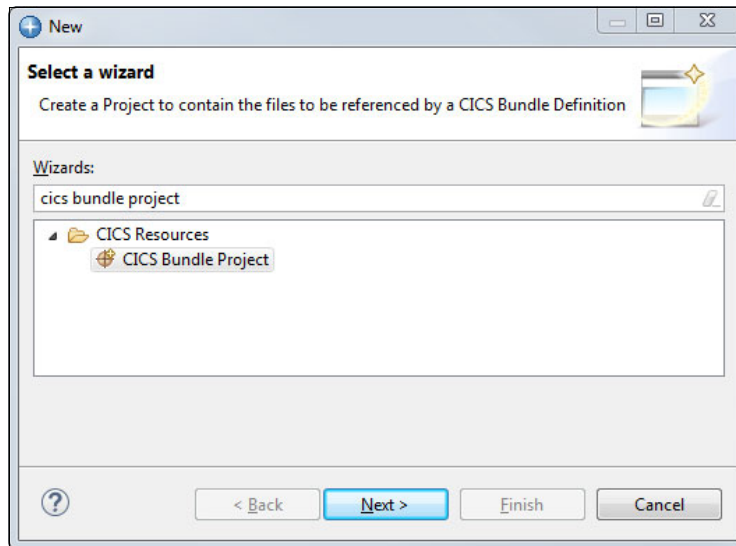


Figure 2-27 Creating a CICS Bundle Project

3. Enter a name for the new project. In our example, we used `com.ibm.cicsdev.restapp2.cicsbundle`, as shown in Figure 2-28.

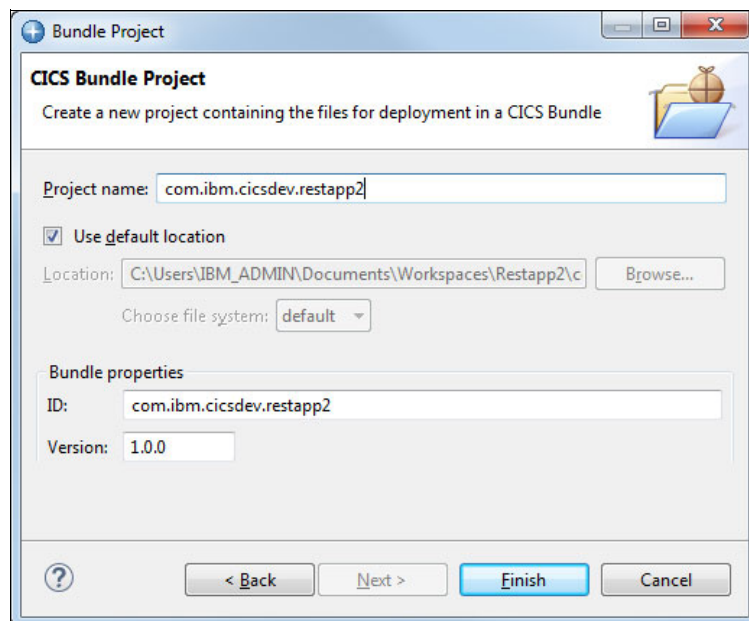


Figure 2-28 Creating a CICS Bundle Project

4. Browse to the folder on your workstation where the Java archive is stored and drag the Java archive into the root of the newly created CICS bundle project in Eclipse.

5. You are prompted to select how files should be imported into the project. Select **Copy files** and click **OK** to proceed.

The CICS bundle project should now show the .war file in the root of the project, as shown in Figure 2-29.

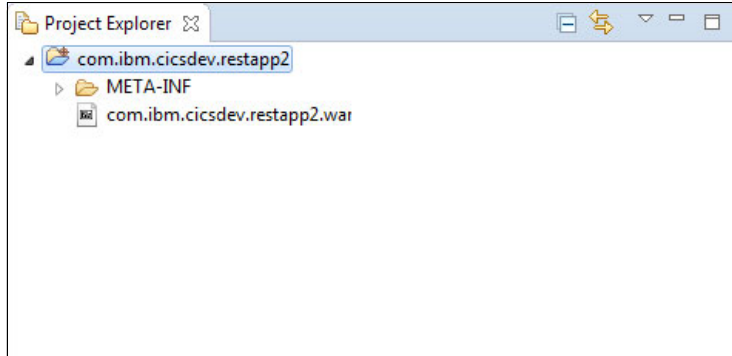


Figure 2-29 CICS bundle project with .war file

6. The .war file can now be added directly to the CICS bundle under Defined Resources by clicking **New** → **Dynamic Web project include**. The current project (com.ibm.cicsdev.restapp2) is listed because it now contains a web artifact, as shown in Figure 2-30.

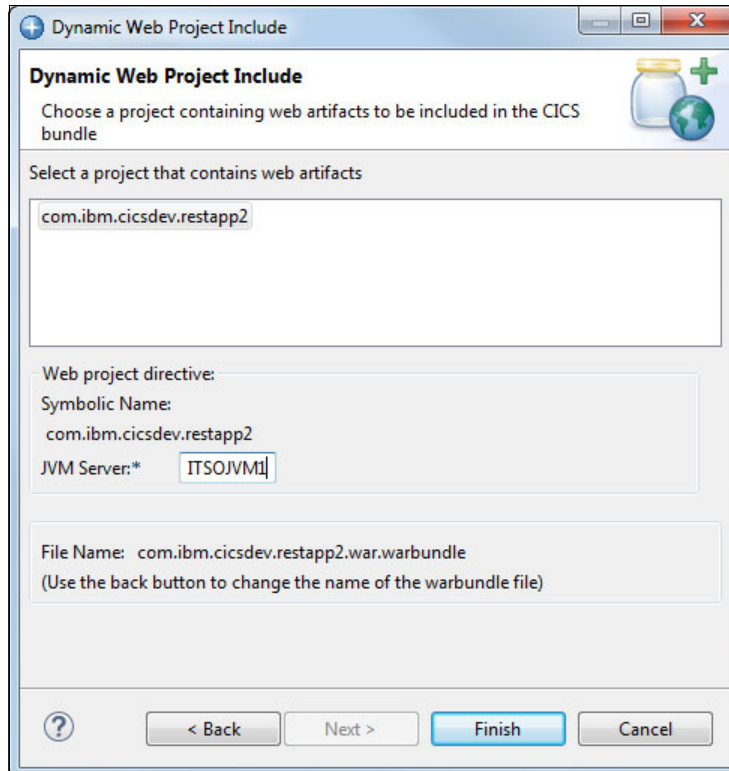


Figure 2-30 CICS bundle project with WAR artifact

7. Select the com.ibm.cicsdev.restapp2 project and add the name of the JVM server, which in our environment is ITSOJVM1.

8. The CICS bundle project now displays the .war file and the .warbundle part file as the two components in the project, as shown in Figure 2-31.

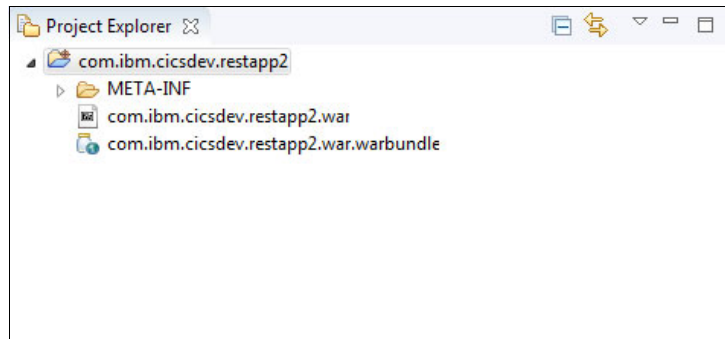


Figure 2-31 CICS bundle project completed

The CICS bundle project is now ready to be exported to zFS by using the same method as described in 2.2.2, “Deployment as an application element in server.xml” on page 37. We exported this project to the following zFS directory location:

```
/var/cicsts/SC8CICS7/ITS0JVM1/bundles/com.ibm.cicsdev.restapp2_1.0.0/
```

9. A CICS BUNDLE resource definition is also needed. We created another CICS BUNDLE that is named RESTAPP2 with the BUNDLEDIR attribute pointing to this bundle directory and installed this BUNDLE into our CICS region.

The Liberty messages.log now shows that this application was installed by using the URL <http://wtsc80:57080/com.ibm.cicsdev.restapp2>, as shown in Example 2-15.

Example 2-15 Installing restapp2

```
11/2/17 11:31:17:120 EDT 000025b5 com.ibm.ws.config.xml.internal.ConfigRefresher
  A CWWKG0016I: Starting server configuration update.
11/2/17 11:31:17:131 EDT 000025b5 com.ibm.ws.config.xml.internal.XMLConfigParser
  A CWWKG0028A: Processing included configuration resource:
/var/cicsts/SC8CICS7/wlp/servers/itsowlp1/restapp.xml
11/2/17 11:31:17:160 EDT 000025b5 com.ibm.ws.config.xml.internal.ConfigRefresher
  A CWWKG0017I: The server configuration was successfully updated in 0.041
seconds.
11/2/17 11:31:17:175 EDT 000027b3 com.ibm.ws.app.manager.AppMessageHelper
  I CWWKZ0018I: Starting application com.ibm.cicsdev.restapp2.
11/2/17 11:31:17:263 EDT 000027b3 com.ibm.ws.webcontainer.osgi.webapp.WebGroup
  I SRVE0169I: Loading Web Module: com.ibm.cicsdev.restapp2.
11/2/17 11:31:17:263 EDT 000027b3 com.ibm.ws.webcontainer
  I SRVE0250I: Web Module com.ibm.cicsdev.restapp2 has been bound to
default_host.
11/2/17 11:31:17:264 EDT 000027b3 com.ibm.ws.http.internal.VirtualHostImpl
  A CWWKT0016I: Web application available (default_host):
http://wtsc80.cpolab.ibm.com:57080/com.ibm.cicsdev.restapp2/
11/2/17 11:31:17:264 EDT 000027b3 com.ibm.ws.app.manager.AppMessageHelper
  A CWWKZ0001I: Application com.ibm.cicsdev.restapp2 started in 0.089 seconds.
```

2.3.4 Pausing and resuming a server

When a web application is deployed or updated into a Liberty JVM server, a period can occur when not all of the dependent resources are available, which results in application level errors. During the update process, a web application also must be removed to allow the new version to be installed. During this period, the Liberty server returns an HTTP 404 response code and a Context Root Not Found error page for any requests to the web application.

In this situation, pausing and resuming the Liberty server HTTP endpoint can allow updates to proceed while the application is taken offline, which prevents application-level errors from occurring during the update. When combined as part of a highly available port sharing cluster, this pausing and resuming the Liberty server can be used to provide a continuously available application environment. For more information, see Chapter 9, “Port sharing and cloning regions” on page 253.

The Liberty server pause function is started by using the Liberty server script, which can be driven from the UNIX System Services command line or BPXBATCH shell by using the CICS-supplied wlpenv script, which is in the JVM server working directory, as shown in Example 2-16.

Example 2-16 Server pause

```
> cd /var/cicsts/SC8CICS7/ITS0JVM1/  
> ./wlpenv server pause itsowlp1
```

```
Executing: /usr/lpp/cicsts/cicsts54/wlp/bin/server pause itsowlp1  
JAVA_HOME=/usr/lpp/java/J8.0_64_SR4  
WLP_INSTALL_DIR=/usr/lpp/cicsts/cicsts54/wlp  
WLP_USER_DIR=/var/cicsts/SC8CICS7/wlp  
WLP_OUTPUT_DIR=/var/cicsts/SC8CICS7/wlp/servers  
SERVER_NAME=itsowlp1
```

```
Pausing the itsowlp1 server.  
Pausing the itsowlp1 server completed.
```

Running the **server pause** command for our Liberty server `itsowlp1` produced the messages that are shown in Example 2-17. These messages indicate that the `defaultHttpEndpoint` was stopped and the `restapp` and `restapp2` applications were removed.

Example 2-17 Liberty messages.log: Server pause

```
[11/14/17 7:08:04:292 EST] 00000027
m.ws.kernel.launch.internal.PauseableComponentControllerImpl I CWWKE0924I: A request was
received to pause the following components in the server: defaultHttpEndpoint
[11/14/17 7:08:04:293 EST] 00000027 com.ibm.ws.tcpchannel.internal.TCPChannel
I CWWK00220I: TCP Channel defaultHttpEndpoint has stopped listening for requests on host
wtsc80.cpolab.ibm.com (IPv4: 9.76.61.131) port 57080.
[11/14/17 7:08:04:293 EST] 00000027 com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0028I: Web application moved (default_host):
https://wtsc80.cpolab.ibm.com:57443/restapp2/
[11/14/17 7:08:04:293 EST] 00000027 com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0028I: Web application moved (default_host):
https://wtsc80.cpolab.ibm.com:57443/com.ibm.cicsdev.restapp/
[11/14/17 7:08:04:295 EST] 00000027 com.ibm.ws.tcpchannel.internal.TCPChannel
I CWWK00220I: TCP Channel defaultHttpEndpoint-ssl has stopped listening for requests on
host wtsc80.cpolab.ibm.com (IPv4: 9.76.61.131) port 52443.
[11/14/17 7:08:04:295 EST] 00000027 com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0017I: Web application removed (default_host):
https://wtsc80.cpolab.ibm.com:57443/restapp2/
[11/14/17 7:08:04:295 EST] 00000027 com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0017I: Web application removed (default_host):
https://wtsc80.cpolab.ibm.com:57443/com.ibm.cicsdev.restapp/
[11/14/17 7:08:04:295 EST] 00000027
m.ws.kernel.launch.internal.PauseableComponentControllerImpl
I CWWKE0938I: A pause request completed.
```

After an application is updated, the Liberty server or a specific endpoint can then be resumed by using the following command:

```
> ./wlpenv server resume itsowlp1
```

Note: In addition to pausing a server, an HTTP endpoint can be started in the paused state by setting the attribute `enabled="false"` on the `httpEndpoint` element. This configuration allows applications to then be made available after server startup by using the **server resume** command.



Link to Liberty

In this chapter, we describe how a CICS program can link to a Java EE application in Liberty in CICS and pass data to and from it by using a channel. This process is shown by using a simple example that [is available from the GitHub website](#).

This chapter includes the following topics:

- ▶ 3.1, “Overview” on page 62
- ▶ 3.2, “Link to Liberty sample application” on page 65
- ▶ 3.3, “Qualities of service” on page 82

3.1 Overview

Link to Liberty is a new CICS Liberty feature that is available in CICS TS v5.3 and later. This feature enables Java methods in Liberty in CICS to be started by using an EXEC CICS LINK, which passes data in CICS containers by using a channel. The Java methods must be in a Plain Old Java Object (POJO), which is packaged as part of a web application.

This feature extends the ability to link to Java programs in a Liberty JVM server, alongside the capability to link to programs in an OSGi JVM server. How Link to Liberty fits into the architecture of the Liberty JVM server in CICS TS is shown in Figure 3-1.

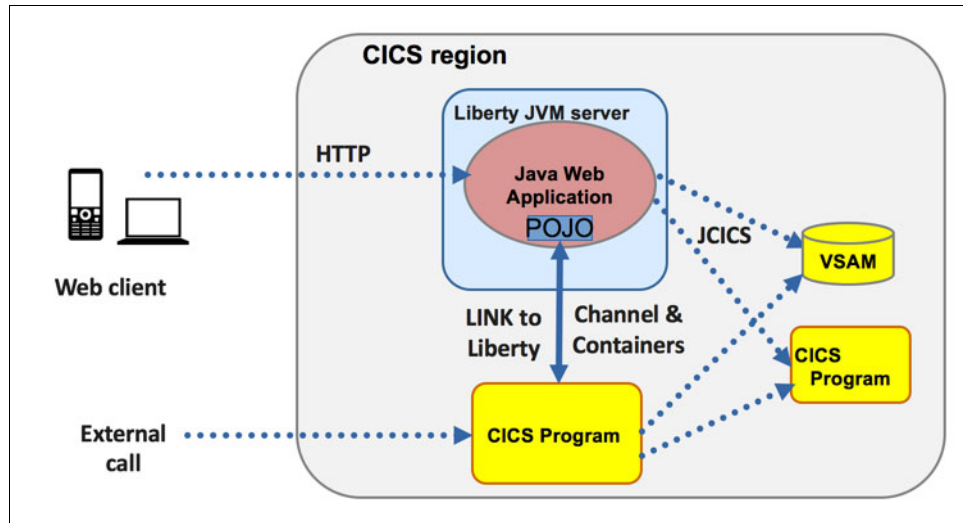


Figure 3-1 Link to Liberty feature architecture

As shown in Figure 3-1, Link to Liberty also provides an indirect means for an external caller to transmit requests to a Java method in a Liberty JVM server.

For more information about Link to Liberty reference documentation, [see IBM Knowledge Center](#).

After a suitable Java method is enabled for Link to Liberty and the web application is started, Link to Liberty can be used as the target of an **EXEC CICS LINK** command as with any other CICS program. It can be started by using a link from non-Java languages because of the initial program of a transaction or by using the **START** and **START CHANNEL CICS API** commands.

You might want to link to a Java EE application from a CICS program for the following reasons:

- ▶ Some reusable Java code is available as part of a web application and you want to link to it from a CICS application as well. By using Link to Liberty as an entry point, you must maintain only a single piece of logic. Your Java code can also access CICS resources by using JCICS APIs.
- ▶ You use Liberty in CICS and you want to simplify your infrastructure by moving other Java applications into Liberty so that you can consolidate the footprint of CICS JVMs. This consolidation reduces the number and variety of JVMs in your environment.
- ▶ You use CICS Java business logic, which currently runs in a CICS OSGi JVM server and you want to port it to run within Liberty in CICS. By using this configuration, you can use it within a web application, while still calling it from a CICS program.

These scenarios require some programming to create new Link to Liberty methods or to rework code to be suitable for Link to Liberty. The calling code also might need to be modified.

A Link to Liberty program also can be used, as shown in the following examples:

- ▶ As the target of a remote ECI call
- ▶ To call from Java to Java within the same CICS region
- ▶ As a target for dynamic program routing

3.1.1 Prerequisites

To use Link to Liberty, the following minimum prerequisites must be met:

- ▶ CICS TS V5.4 or CICS TS V5.3 with APAR PI63005 applied
- ▶ A configured Liberty JVM server that is running in integrated mode
- ▶ CICS Explorer or CICS build toolkit V5.3.0.8, or later

We used CICS TS V5.4 on z/OS 2.3 that is running Java 1.8 SR4, and IBM CICS Explorer Version 5.4.2.

3.1.2 How it works

In this section, we describe how to develop, deploy, and run Link to Liberty applications and what occurs at each stage in the process.

Developing Link to Liberty applications

Your Java code must be a Plain Old Java Object (POJO), which is packaged in WAR or an EAR. It makes sense to call Java code that implements business logic rather than web layers, such as servlets or JAX-RS resource classes. If you want to call an EJB, you must create a POJO wrapper that CICS can call.

For more information about restrictions on the Java methods that are eligible as targets for Link to Liberty, [see IBM Knowledge Center](#).

Because of these restrictions, CICS cannot pass any arguments to any method (including Constructors), which is the target of a Link to Liberty call. Also, the method might not include a return value; therefore, it is declared as void.

In practice, it is unlikely that you can use the methods in Java code as Link to Liberty targets by adding CICSProgram annotations. However, you can easily create Link to Liberty entry point methods, which then start methods and pass appropriate arguments.

Link to Liberty methods can be in new or existing Java classes, if the eligibility criteria are satisfied. However, it likely makes more sense to create Java classes specifically to contain the Link to Liberty entry points so that the CICS-specific code (which deals with input and output containers, data marshalling, and so on) is all encapsulated and kept separate from Java web application logic.

Note: To use Link to Liberty, you likely must create methods for the Link to Liberty endpoints, and encapsulate the methods within a Link to Liberty-specific class.

@CICSProgram annotation

After you decide on the method you want to call by using Link to Liberty, add an @CICSProgram annotation to the method, as shown in Example 3-1.

Example 3-1 An @CICSProgram annotation example

```
@CICSProgram("GETSUPPI")
public void getSupplierInfo() throws CicsConditionException { }
```

If you use the CICS Explorer, validation occurs automatically to ensure that your annotation is correctly positioned and that the method that it annotates is eligible for Link to Liberty.

The annotation must be on a suitable method. The method must conform to the rules for eligible Link to Liberty methods. The annotation also must include a value attribute of a PROGRAM name. This name must be a Java String and can be a constant, such as a static final variable.

The process of annotating and building Link to Liberty methods is shown in Figure 3-2.

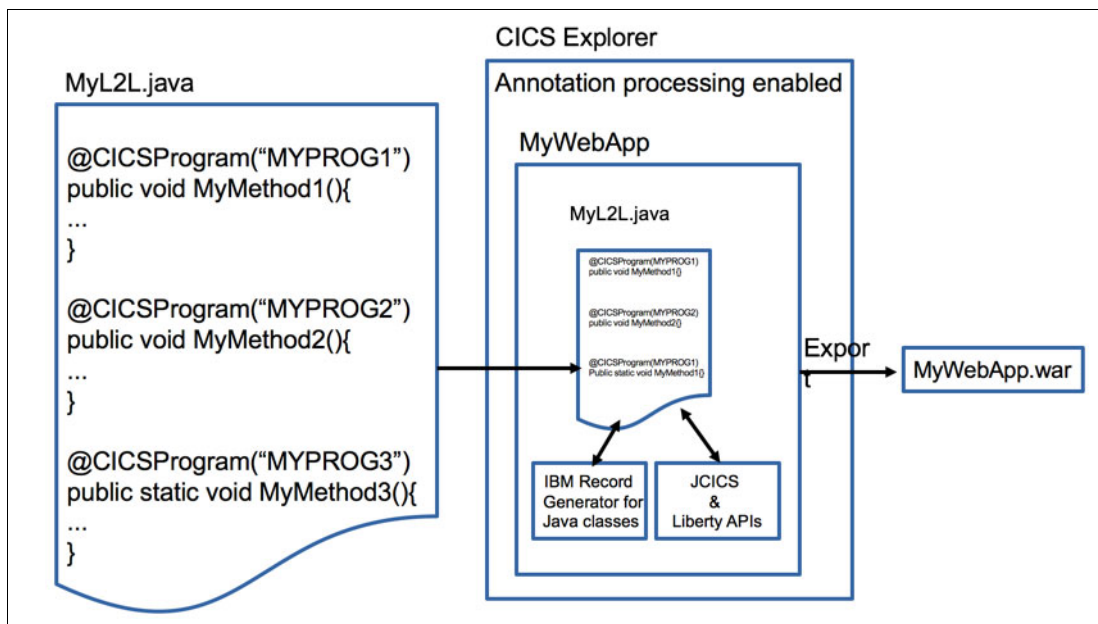


Figure 3-2 Annotation and build process for Link to Liberty

The Java class that contains the annotated Link to Liberty methods is included in a web application with annotation processing enabled. When exported as a WAR, the methods feature associated artifacts that identify them to CICS as Link to Liberty methods. Therefore, they are made available as linkable programs when they are installed into the Liberty JVM server.

Invoke

When deployed into Liberty, the Link to Liberty method can be started as a linkable CICS program (as with any other CICS program) and passes a channel to be used for input and output containers. Link to Liberty calls the method with the corresponding @CICSProgram annotation and passes the channel in the environment, as shown in Figure 3-3 on page 65.

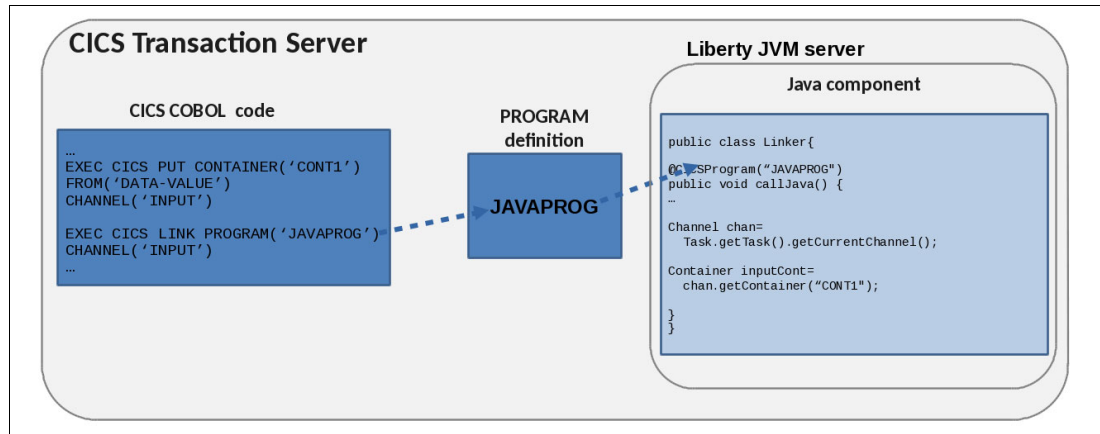


Figure 3-3 Link to Liberty runtime processing.

The caller populates input containers in a channel and then links to the generated PROGRAM definition that is passing the channel. The target Java method receives control and can access the data in the container and create response containers in the channel, if required. At the end of the method, control returns to the calling program, as it does for a normal CICS LINK.

3.2 Link to Liberty sample application

In this section, we describe the process of developing a simple Link to Liberty application. This process includes annotating the code, building the web application, deploying to Liberty in CICS, and calling it by using a link from a COBOL terminal program.

As a starting point to demonstrate this process, we use the `LinkToLiberty.java` example that is included as part of the `restapp-ext` sample from the GitHub `cicsdev` organization.

3.2.1 Building the sample application

The procedure to download and build the `restapp-ext` sample is similar to that process that is used for the `restapp` sample documented in Chapter 2, “Deploying a web application” on page 27. A few minor differences exist between the processes (those differences are highlighted in this section).

Complete the following steps:

1. Download the `restapp-ext` sample [from the GitHub website](#):
 - a. Click **Clone or download** → **Download ZIP**.
 - b. When the repository download completes, decompress the file to a suitable directory on your workstation.
2. Import the Java source:
 - a. By using the CICS Explorer development environment and the Java EE perspective, create a dynamic web project that is named `com.ibm.cicsdev.restappext` and add the Java samples to the `src` folder.

Tip: The Java source files can be dragged directly from the repository subdirectory `/src/Java` instead of using the import process in this step.

- b. The Java samples are in the repository subdirectory `/src/Java`. Add these samples to the `src` folder.
- c. Expand the dynamic web project `com.ibm.cicsdev.restappext`.
- d. Expand Java Resources, right-click **src**, and then, select **Import**.
- e. In the Import window, expand **General** and select **File System**. Then, click **Next**.
- f. In the File System Import window, click **Browse** to choose a From directory for the import and browse to select the `src/Java` subdirectory of the decompressed repository `cics-java-liberty-restapp-ext-master` directory, as shown in Figure 3-4.

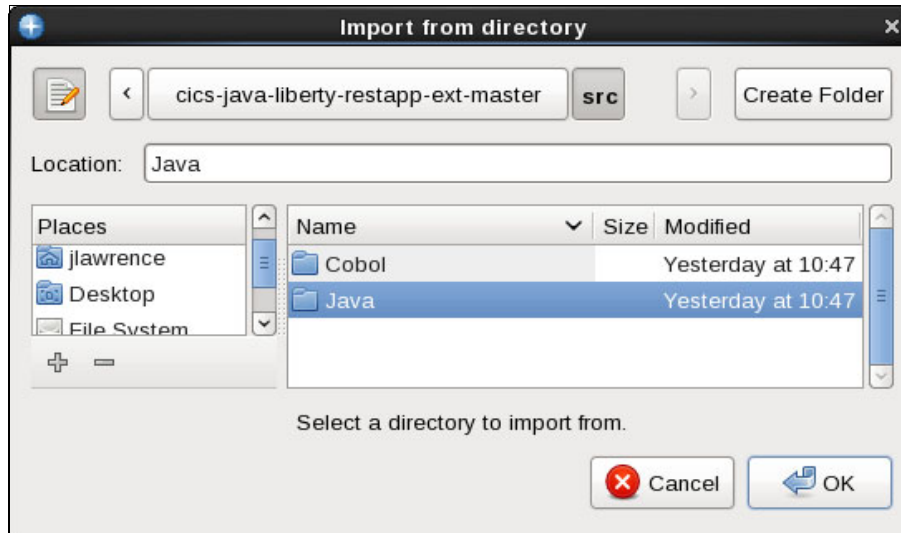


Figure 3-4 Selecting the `src/Java` directory from which to import

- g. Click **OK** to return to the File System Import window.
- h. Expand the `src/Java` folder and select the **com** directory, as shown in Figure 3-5.

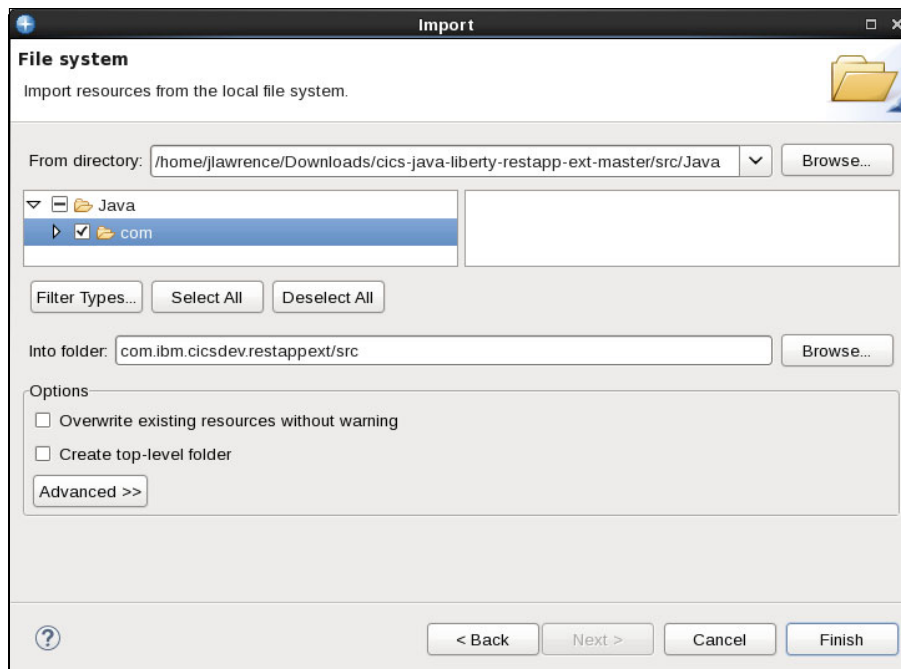


Figure 3-5 Importing the `com` subdirectory into `/src`

- i. Click **Finish** to complete the import process. Although the Java source files are imported into the project /src folder, build errors occur, which are corrected next. Other methods are available to complete the same import; however, it is important that the com directory and its content are imported into /src.

Note: The restapp-ext sample includes several sample Java programs, which show the use of the JCICS API in Java. Because only the LinkToLiberty.java source file is needed in package com.ibm.cicsdev.restappext, all of the other Java content can be deleted from the project. However, the other sample files in the project can be retained.

3. Add the CICS Liberty JVM server libraries to the build path:
 - a. Right-click the dynamic web project and select **Build Path** → **Configure Build Path**.
 - b. Select the **Libraries** tab of the Java Build Path and click **Add Library**.
 - c. Select library type **CICS with Java EE and Liberty**, as shown in Figure 3-6.

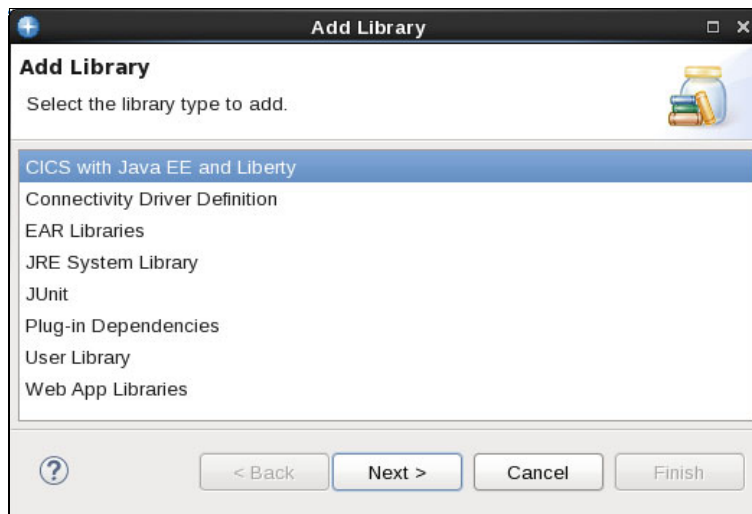


Figure 3-6 Select CICS with Java EE and Liberty library type

- d. Click **Next**.
- e. Select **Version CICS TS 5.4** to deploy your project to, and click **Finish**. In the Java Build Path window, a new entry for CICS with Java EE and Liberty Library is shown (see Figure 3-7).

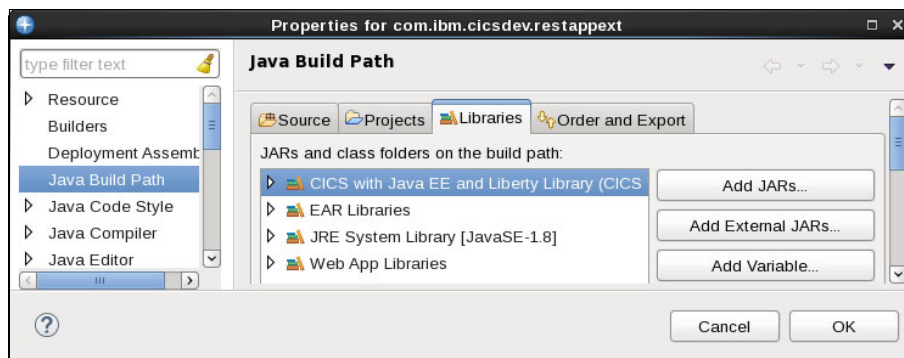


Figure 3-7 Adding CICS with Java EE and Liberty Library to the project build path

- f. Click **OK**. The project is rebuilt and the number of errors is reduced, although some errors remain. The process that is used to correct this issue is described next.
4. Import the IBM Record Generator for Java generated jar file.

Many of the Java samples in `restapp-ext`, including `LinkToLiberty.java`, use JCICS to transmit structured data between Java and native CICS resources, such as programs, files, or temporary storage queues.

The `restapp-ext` sample includes a pre-built Java archive file that contains IBM Record Generator for Java-generated helper classes to marshal data for the data structures that are used in the sample. This Java archive file must be imported into the web application project so that these classes are available.

Tip: The `com.ibm.cicsdev.restappext.generated.jar` file can be dragged directly from the repository subdirectory `/lib` into `/WebContent/WEB-INF/lib`, instead of using the wizard import process in this step.

Complete the following steps to copy the `com.ibm.cicsdev.restappext.generated.jar` file to the folder `/WebContent/WEB-INF/lib` relative to the root of your dynamic web project:

- Expand the dynamic web project and click **WebContent** → **WEB-INF**.
- Right-click `/lib` and select **Import**.
- Click **General** → **File System**. Click **Next**.
- Click the **Browse** button, and browse to the decompressed `restapp-ext` directory. Select the `/lib` subdirectory, as shown in Figure 3-8.

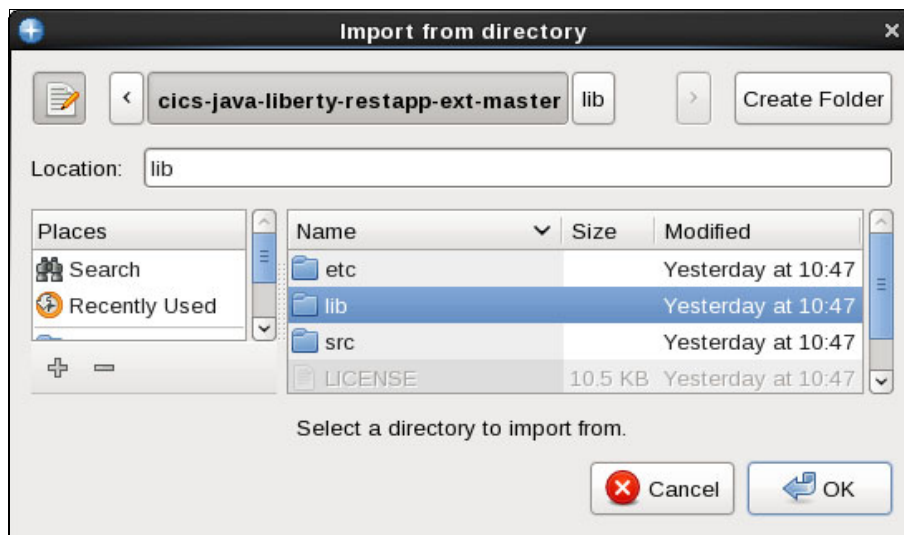


Figure 3-8 Browsing to the repository directory and selecting `/lib`

- e. Click **OK**. In the File System Import window, select **com.ibm.cicsdev.restappext.generated.jar**, as shown in Figure 3-9.

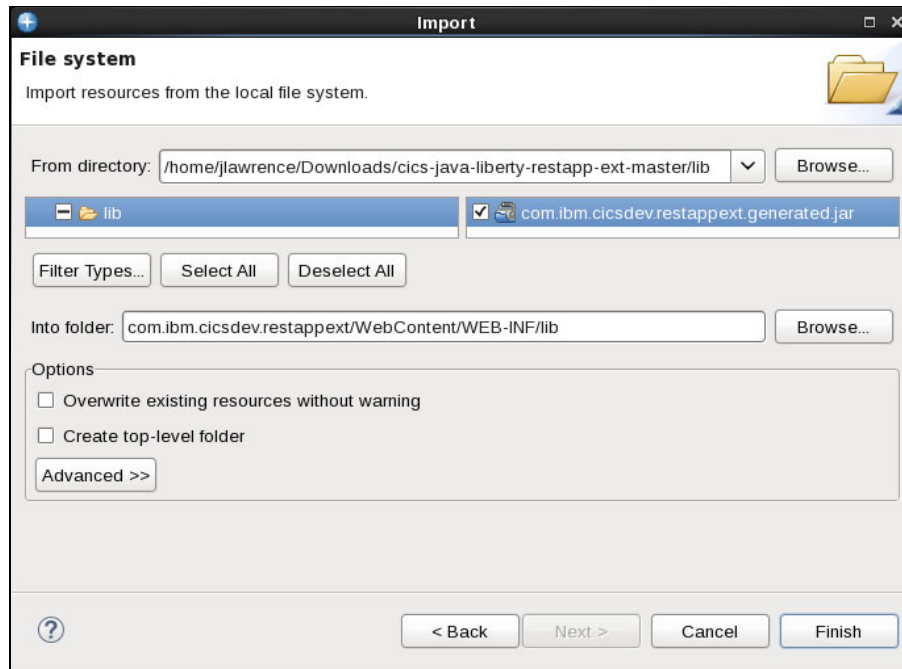


Figure 3-9 Selecting the generated Java archive and import into /WEB-INF/lib

- f. Click **Finish** to complete the import.

The `com.ibm.cicsdev.restappext.generated.jar` file is added automatically to the project build path as a Web App Library. The remaining build errors in the project should be resolved.

5. Enable annotations for the project.

Eligible Java methods are enabled for Link to Liberty by using the `CICSProgram` annotation on the method. This method specifies the name of the CICS program to be created when the method is installed. For this process to work, annotation processing must be enabled for the dynamic web project. A Java compiler warning against `LinkToLiberty.java` might be generated, which results from a warning on the `@CICSProgram` annotation on the `getSupplierInfo()` method, as shown in Figure 3-10.

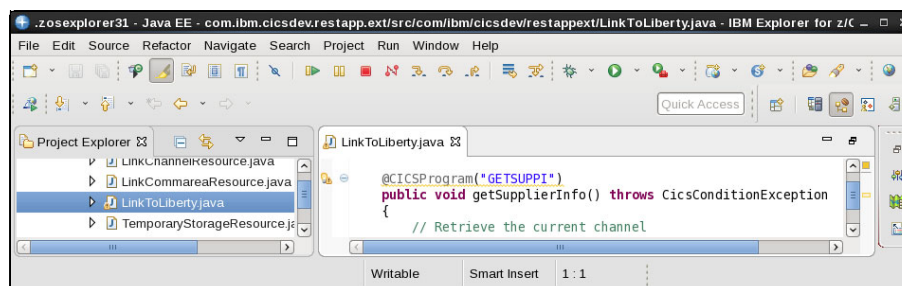


Figure 3-10 Warning flag in CICS Explorer because annotations are not enabled

Complete the following steps:

- a. Right-click the project and select **Properties**.
- b. Locate and expand the Java Compiler category and select **Annotation Processing**. If necessary, select the **Enable project specific settings** option.
- c. Select both of the Enable annotation options, as shown in Figure 3-11.

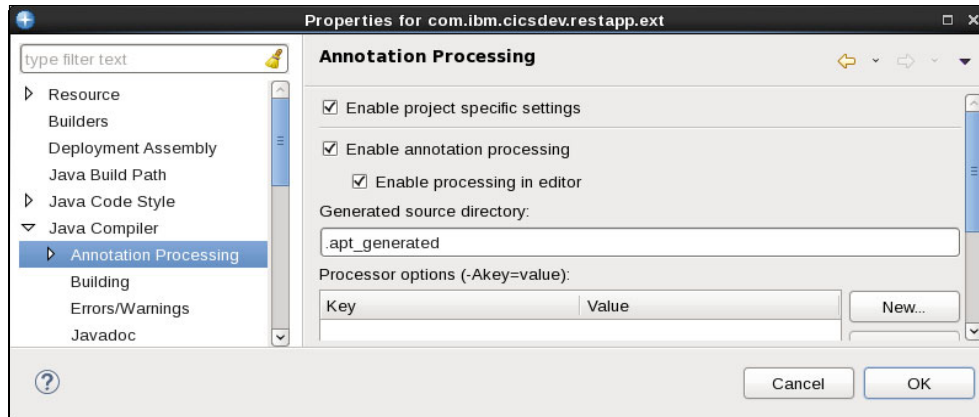


Figure 3-11 Enable annotation processing in the dynamic web project properties

- d. Click **OK** to complete the update. The warning on the @CICSProgram annotation should be resolved, as shown in Figure 3-12.

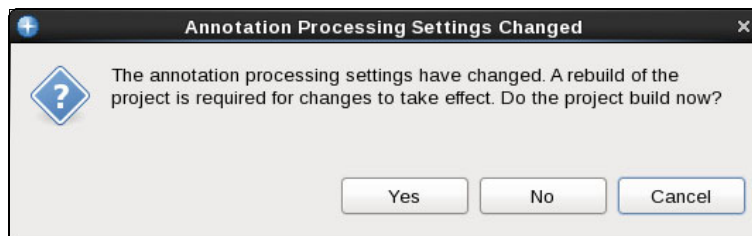


Figure 3-12 Annotation Processing Settings Changed window

- e. Click **Yes** to rebuild the project after enabling the annotations.

An alternative method to enable annotations in the project is to hover over the warning that is next to the annotation in the Java source editor for the Link to Liberty method and use the Enable annotation processing quick fix. For more information, see the [Link to Liberty now available in CICS TS V5.3 page](#) of the IBM developerWorks® website.

Important: Ensure that you enable annotations for the dynamic web project. Although this enablement might generate only a warning in the development environment, the Link to Liberty program is not available when deployed.

The restrictions for method eligibility for Link to Liberty enablement are checked in the CICS Explorer development environment. For example, we made a copy of `LinkToLiberty.java` and introduced an eligibility problem by adding a `String` argument to the method, which is not permitted for Link to Liberty. CICS Explorer reported this issue as an error (see Table 3-1).

Table 3-1 Example annotation problem reported by CICS Explorer for `@CICSProgram`

Description	A method with the annotation <code>@CICSProgram</code> must have no arguments
Resource	<code>LinkToLiberty2.java</code>
Path	<code>/com.ibm.cicsdev.restappext/src/com/ibm/cicsdev/restappext</code>
Location	Line 40
Type	Annotation Problem (Java 6 processor)

This issue appears in the CICS Explorer, editor as shown in Figure 3-13.

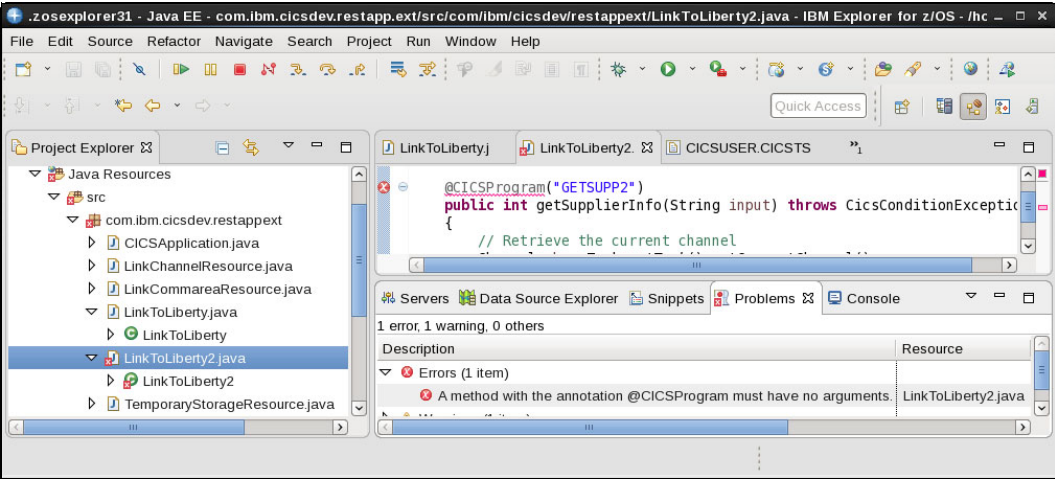


Figure 3-13 CICS Explorer reports methods that are not eligible to be enabled for Link to Liberty

Note: A Link to Liberty method should not have a return value. As shown in Figure 3-13, the return type is `void` instead of `int`, although this issue is not flagged.

6. Check the Java compiler target level.

Ensure that the web project is targeted to compile at a level that is compatible with the Java level that is used on CICS. Specifically, the Java Compiler compliance level must be less than or equal to the Java level of the Liberty JVM server.

Complete the following steps:

- Right-click the dynamic web project and select **Properties**.
- Select **Java Compiler** and review the JDK Compliance settings. We enabled project-specific settings and chose to use the compliance from execution environment (which was 1.8), as shown in Figure 3-14 on page 72.

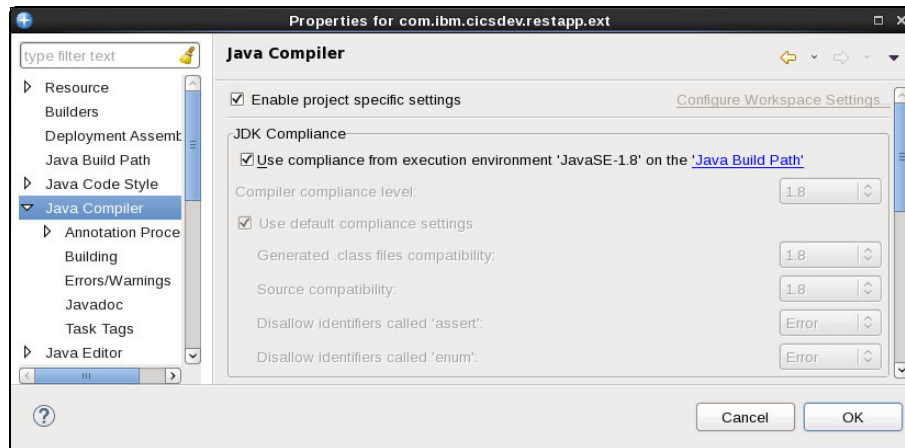


Figure 3-14 Reviewing Java Compiler compliance settings

A suitable compiler compliance level can be achieved by setting it at the workspace level, or by using project-specific settings.

c. Click **OK** to confirm any changes and close the Properties window.

7. Create a CICS bundle project for deployment.

This step is optional and is required only if you plan to use the CICS Bundle deployment method for the application. For more information about deployment methods and how to create a CICS bundle project, see Chapter 2, “Deploying a web application” on page 27.

If you want to deploy as a CICS bundle, create a CICS bundle project that is named `com.ibm.cicsdev.restappext.cicsbundle` and add a dynamic web project include for project `com.ibm.cicsdev.restappext`, as shown in Figure 3-15.

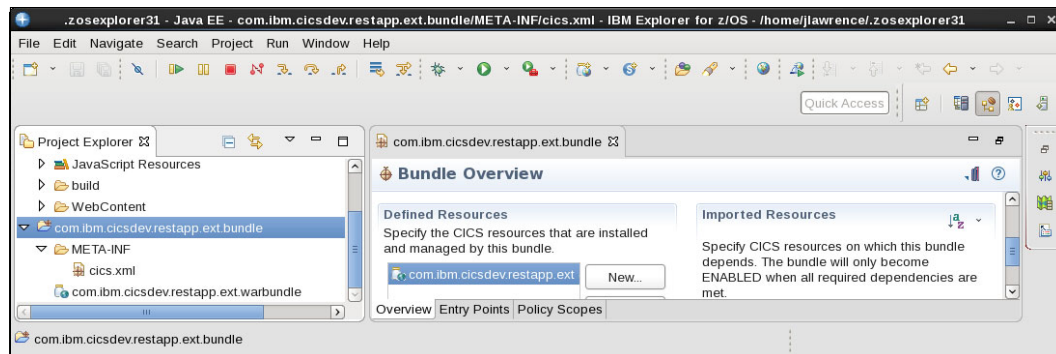


Figure 3-15 CICS Bundle definition including the dynamic web project

3.2.2 Sample application

The sample `LinkToLiberty.java` class that is shown in Example 3-2 is a simple POJO. It shows how the `@CICSProgram` annotation can be used to enable a Java method for Link to Liberty.

Example 3-2 Annotated Java source for the `LinkToLiberty.java` sample

```
import com.ibm.cics.server.Channel;           1
import com.ibm.cics.server.CicsConditionException;
import com.ibm.cics.server.Container;
import com.ibm.cics.server.Task;
```

```

import com.ibm.cics.server.invocation.CICSProgram;           2
import com.ibm.cicsdev.restappext.generated.StockPart;      3
import com.ibm.cicsdev.restappext.generated.Supplier;       4
public class LinkToLiberty
{
    @CICSProgram("GETSUPPI")                                  5
    public void getSupplierInfo() throws CicsConditionException 6
    {
        Channel ch = Task.getTask().getCurrentChannel();      7

        Container contStockPart = ch.getContainer("STOCK-PART"); 8

        StockPart sp = new StockPart( contStockPart.get() );    9

        int iSupplierId = sp.getSupplier();                     10

        Supplier supplier = new Supplier();                     11

        supplier.setSupplierId(iSupplierId);                    12

        String name = "Supplier #" + iSupplierId;              13
        supplier.setSupplierName(name);

        Container contSupplier = ch.createContainer("SUPPLIER");14
        contSupplier.put( supplier.getByteBuffer() );           15
    }
}

```

The main features of the code are highlighted by the following numbered annotations that are shown in Example 3-2 on page 72:

1. Imports for JCICS classes.
2. Import that is required for the @CICSProgram annotation.
3. Imports for the IBM Record Generator for Java generated marshalling classes.
4. The LinkToLiberty class is concrete and includes a default constructor.
5. @CICSProgram annotation specifies the program name GETSUPPI.
6. getSupplierInfo() is the target method for the GETSUPPI PROGRAM definition.
7. Get the Channel passed on the LINK.
8. Get the input Container that is named STOCK-PART.
9. Get the content of STOCK-PART and create an IBM Record Generator for Java helper class.
10. Use the helper class to extract the Supplier ID from the container data.
11. Create a Supplier helper to construct the response container.
12. Use the helper class to set the SupplierId field in the response.
13. Use the helper class to set the SupplierName field in the response.
14. Put the structured response data into a response Container SUPPLIER.
15. Return to caller at the end of the method; equivalent to CICS RETURN.

When included in a web application and deployed in Liberty, a PROGRAM definition for GETSUPPI is created. This creation then allows a LINK call to GETSUPPI, which starts the getSupplierInfo() method and passes a channel for input and output.

3.2.3 Deploying the sample

After the dynamic web application project is cleanly built in the development environment and a CICS bundle project is created (if needed), it can be deployed into Liberty by any of the deployment methods that are described in Chapter 2, “Deploying a web application” on page 27.

Adding features to the Liberty server

Add `<feature>cicsts:link-1.0</feature>` to the `featureManager` tag in `server.xml`. This addition must be done before a web application is deployed that contains Link to Liberty programs for them to be made available by using program link.

Add the security feature `<feature>cicsts:security-1.0</feature>` to the Liberty server, if required. If you use CICS security and your application requires the Java subject to be available when started by using Link to Liberty, you must add the CICS security Liberty feature to the `featureManager` section in `server.xml`, as shown in Example 3-3.

Example 3-3 The `cicsts:link-1.0` and `cicsts:security-1.0` features in `server.xml`

```
<featureManager>
  <feature>cicsts:core-1.0</feature>
  <feature>cicsts:link-1.0</feature>
  <feature>cicsts:security-1.0</feature>
</featureManager>
```

The Java Subject might be required to be available for the following reasons:

- ▶ You are starting EJBs from your application and they use EJB role security authorization.
- ▶ Other application code that you are starting by using Link to Liberty include a dependency on the Java Subject being present; for example, if you use Java EE role-based security.

Adding the `cicsts:link-1.0` feature to Liberty results in messages, such the message in the Liberty `messages.log` file that is shown in Example 3-4.

Example 3-4 `messages.log` output when the Link to Liberty feature is installed

```
A CWWKG0017I: The server configuration was successfully updated in 2.532 seconds.
I J2CA7018I: Installing resource adapter com.ibm.cics.wlp.program.link.connector.
A CWWKF0012I: The server installed the following features: [cicsts:link-1.0,
jaxb-2.2, jcaInboundSecurity-1.0].
A CWWKF0008I: Feature update completed in 2.600 seconds.
A J2CA7001I: Resource adapter com.ibm.cics.wlp.program.link.connector installed in
0.378 seconds.
```

Important: Ensure that you add the `cicsts:link-1.0` feature to Liberty before you deploy the Link to Liberty methods. Although no warning or message is provided if this feature is not added, the Link to Liberty methods are unavailable when deployed.

Deploying the sample application

For more information about deployment options for the web application, see Chapter 2, “Deploying a web application” on page 27. Any of the deployment methods can be used for applications that contain Link to Liberty methods.

Deployment

Installing a web application into a suitably configured Liberty JVM server with the `cicsts:link-1.0` feature installed triggers CICS to examine the web archive. It also identifies any artifacts that are generated by the annotation processor as a result of the `@CICSProgram` annotation.

In the Liberty `messages.log` and the CICS job log, the usual messages that are associated with Liberty application deployment are observed. In addition, the messages that are shown in Example 3-5 are written to the CICS `MSGUSR` log.

Example 3-5 CICS messages when deploying a Link to Liberty method.

```
DFHPG0101 10/26/2017 10:07:51 SC8CICS7 CICSUSER ???? Resource definition for  
GETSUPPI has been added.
```

```
DFHSJ1204 10/26/2017 10:07:51 SC8CICS7 A linkable service has been registered for  
program GETSUPPI in JVMSERVER ITS0JVM1 with  
classname com.ibm.cicsdev.restappext.LinkToLiberty, method getSupplierInfo.
```

These messages indicate that a program resource for GETSUPPI was dynamically created and that the Link to Liberty method was registered as a linkable service.

CICS dynamically creates a `PROGRAM` resource for each `@CICSProgram` annotated method in the application, unless a `PROGRAM` resource with the same name exists. If a `PROGRAM` with the same name is installed at the point when the Liberty application is enabled, CICS does not attempt to create a `PROGRAM`. Instead, it validates that the attributes of the `PROGRAM` are suitable and issue a DFHSJ1208 warning to `MSGUSR` if the attributes are not suitable, as shown in Example 3-6.

Example 3-6 CICS messages issued when an unsuitable PROGRAM definition exists

```
DFHSJ1208 10/26/2017 15:56:53 SC8CICS7 An existing definition has been installed  
for PROGRAM GETSUPPI. It is not suitable for use with a linkable service because  
it does not have JVM(YES). The expected value is YES, the value found is NO.
```

```
DFHSJ1204 10/26/2017 15:56:53 SC8CICS7 A linkable service has been registered for  
program GETSUPPI in JVMSERVER ITS0JVM1 with classname  
com.ibm.cicsdev.restappext.LinkToLiberty, method  
getSupplierInfo.
```

You can find these created PROGRAM resources by using CEMT or in the programs view in CICS Explorer and filter on the name of the JVM server. The CICS Explorer view of the created GETSUPPI PROGRAM definition is shown in Figure 3-16. The results are filtered to show the JVM Server and Service Name attributes.

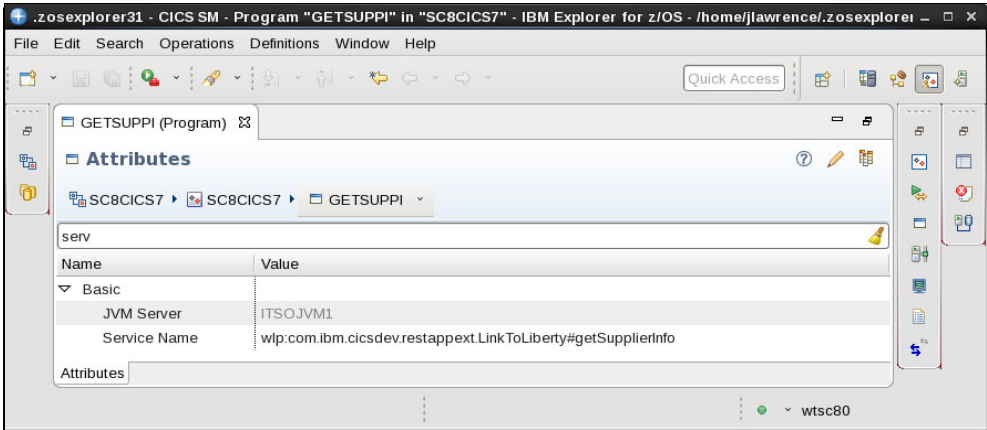


Figure 3-16 CICS Explorer view of the dynamically generated GETSUPPI program

Looking at the same definition in CEMT, we see the GETSUPPI program, as shown in Figure 3-17.

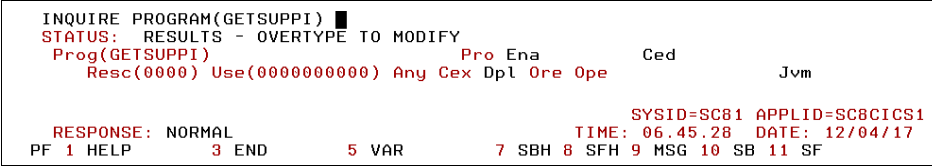


Figure 3-17 Program GETSUPPI created definition view in CEMT

Expanding the PROGRAM definition in CEMT displays attribute information. Attributes of particular interest are listed in Table 3-2.

Table 3-2 Selected attributes of the automatically created GETSUPPI PROGRAM definition

Attribute	Value
Status	Enabled
Cedfstatus	Cedf
Executionset	Dplsubset
Runtime	Jvm
Jvmclass	wlp.com.ibm.cicsdev.restappext.LinkToLiberty#getSupplierInfo
Jvmserver	ITSOJVM1
Installusrid	CICSREGN
Installagent	Dynamic
Definesource	WLPAPP

The structure of the service name (Jvmlclass), which is assembled from four elements, is listed in Table 3-3.

Table 3-3 Jvmlclass attribute for a Link to Liberty PROGRAM definition

Element	Description	Value
Prefix	Constant "wlp:"	wlp:
Package	Java package of the class containing the Link to Liberty method.	com.ibm.cicsdev.restappext
Class	Java class name containing the method to be invoked by LINK.	LinkToLiberty
Method	Specific method to invoke.	getSupplierInfo

Because of the constraints on methods to be eligible for Link to Liberty, service name is sufficient to uniquely identify the Java entry point to be started when linked as a CICS program.

If you construct manually predefined PROGRAM definitions for Link to Liberty methods, the Jvmlclass (service name) must be constructed by following the same structure.

Note: If the deployment of a Link to Liberty application fails, no PROGRAM definitions are created, and no messages are issued, check the following two configuration issues in the first instance:

- ▶ That annotations are enabled in the development environment when the application is built.
- ▶ The cics:link-1.0 feature is included in the feature manager section in Liberty.

In these cases, no other indication of the cause of the deployment problem is available.

If more than one method in an application specifies the same Link to Liberty program name, or if another Link to Liberty method with the same name is installed, the method is not registered and message DFHSJ1205 is issued, as shown in Example 3-7.

Example 3-7 Message issued for a duplicate Link to Liberty program name

```
DFHSJ1205 10/27/2017 11:58:12 SC8CICS7 A linkable service was not registered for
program GETSUPPI in JVMSERVER ITS0JVM1 because another linkable service is already
registered with that program name. The class name was
com.ibm.cicsdev.restappext.LinkToLiberty, method name getSupplierInfo
```

If a PROGRAM definition is installed by using program autoinstall or RDO (which matches the name on the @CICSProgram annotation), no dynamic programs are created. A DFHSJ1208 warning message is issued to MSGUSR if the definition does not match the Link to Liberty method. Example messages are shown in Example 3-8.

Example 3-8 Example messages

JVM(NO)

```
DFHSJ1208 10/26/2017 15:56:53 SC8CICS7 An existing definition has been installed
for PROGRAM GETSUPPI. It is not suitable for use with a linkable service because
it does not have JVM(YES). The expected value is YES, the value found is NO.
```

Incorrect JVMclass

DFHSJ1208 10/26/2017 16:35:55 SC8CICS7 An existing definition has been installed for PROGRAM GETSUPPI. It is not suitable for use with a linkable service because it does not specify the correct JVMCLASS. The expected value is wlp.com.ibm.cicsdev.restappext.LinkToLiberty#getSupplierInfo, the value found is wlp.com.ibm.cicsdev.restappext.LinkToLiberty#getSupplierInfox.

Incorrect JVMserver

DFHSJ1208 10/26/2017 17:22:17 SC8CICS7 An existing definition has been installed for PROGRAM GETSUPPI. It is not suitable for use with a linkable service because it does not specify the correct JVMSERVER. The expected value is ITS0JVM1, the value found is ITS0JVMX.

Equivalent messages are also written to the Liberty dfhjmerr, dfhjmtrc, and messages.log for each of these cases, as shown in Example 3-9.

Example 3-9 Equivalent messages

```
[err] 10/26/2017 17:22:17.373000 UTC E [RUN_SERVICE_Thread-134237]
[com.ibm.cics.wlp.link.impl] [ProgramGenerator] @Error: validateExistingProgram()
- Existing definition installed for PROGRAM GETSUPPI with the wrong JVMSERVER
name. Found 'ITS0JVMX' but expected 'ITS0JVM1'. Problem encountered processing
Bundle RESTAPPX.
```

3.2.4 Running the sample

Our GETSUPPI program is now available to be started by using a program LINK call from another CICS program. A sample LINK2SUP COBOL program is provided in the restapp-ext GitHub repository to start the GETSUPPI Link to Liberty program.

LINK2SUP COBOL program

The provided sample LINK2SUP program is a simple CICS terminal program that shows how to start a Java Link to Liberty method from COBOL. It passes data to and from the method in CICS containers.

It performs the following steps:

1. Reads the terminal input and extracts a numeric supplier ID, if present.
2. Copies the Supplier ID (or CICS task number) into the STOKPART data structure and writes it to CICS container STOCK-PART.
3. Makes the EXEC CICS LINK call to program GETSUPPI and passes the channel with the input container.
4. Checks the response from the link, issues an error message to the terminal, and returns if unsuccessful.
5. If the link succeeded, it reads the output container SUPPLIER from the channel.
6. Constructs a response message to the terminal with some data from the SUPPLIER container.
7. Returns to CICS and end transaction.

Populating input containers, making the EXEC CICS LINK call, and extracting data from response containers often must be done by any CICS program to start a Link to Liberty method in Liberty.

The Link to Liberty method uses JCICS to read from and put to the containers in the channel, which is passed between native CICS and the Liberty environment on the Link to Liberty call.

Compiling the LINK2SUP COBOL program

The LINK2SUP program source is provided in the /src/Cobol subdirectory of the downloaded GitHub repository as LINK2SUP.cbl. Also provided are the two required copybooks STOKPART.cpy and SUPPLIER.cpy.

Upload these source files to appropriate data set members on z/OS and compile LINK2SUP into a load library in DFHRPL or a dynamic load library as normal. You might want to compile with the ADATA option to regenerate the IBM Record Generator for Java-generated data marshalling Java classes that are used in LinkToLiberty.java.

Defining program and transaction for LINK2SUP

Define a transaction and program for LINK2SUP. Suitable DFHCSDUP input definitions for these resources are included in the /etc/DFHCSD.txt in the cics-java-liberty-restapp-ext GitHub repository, as shown in Example 3-10.

Example 3-10 Defining transaction and program for LINK2SUP

```
DEFINE PROGRAM(LINK2SUP) GROUP(RESTAPPX)
    CONCURRENCY(THREADSAFE) DATALOCATION(ANY)
DEFINE TRANSACTION(JL2L) GROUP(RESTAPPX)
    PROGRAM(LINK2SUP) TASKDATALOC(ANY)
```

We included these definitions in the CICS Bundle that we created for the Link to Liberty web application, as shown in Figure 3-18.

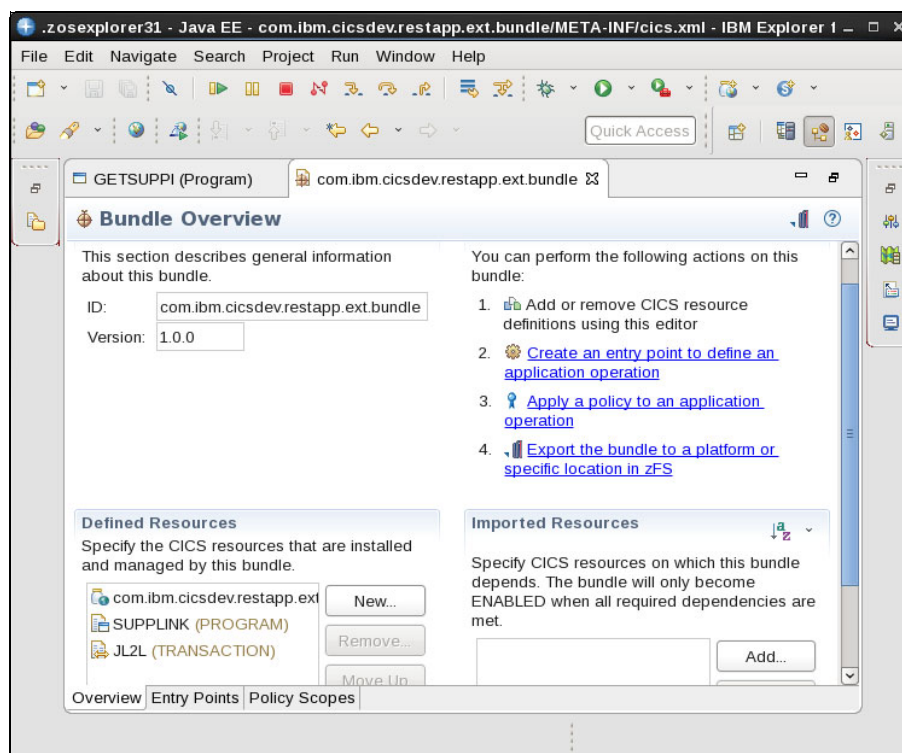


Figure 3-18 CICS Bundle Overview with definitions for LINK2SUP and JL2L

Running the JL2L transaction

At a CICS terminal, enter transaction JL2L followed by a numeric ID of 8 digits or less and press Enter, as shown in the following example:

```
JL2L 12345
```

The following response line is displayed:

```
SUPPLIER ID: 00012345 SUPPLIER NAME: Supplier #12345
```

The supplier name is constructed in the Java method from the input ID.

If you enter only the transaction name JL2L, a response line similar to the following example is displayed:

```
SUPPLIER ID: 00000315 SUPPLIER NAME: Supplier #315
```

The Supplier ID is taken from the caller's EIBTASKN if it is not specified in the terminal input.

3.2.5 Manual program definition

PROGRAM definitions can be installed and defined for Link to Liberty by means other than dynamic creation for the following reasons:

- ▶ If PROGRAM autoinstall is active on the CICS region, an attempt to Link to Liberty before the Link to Liberty program is dynamically created fails. However, a conventional PROGRAM definition is automatically installed during that process, which then blocks dynamic creation for the same PROGRAM name when the Link to Liberty application is installed.
- ▶ If you want to modify other attributes of the PROGRAM definition, such as EDF enablement.
- ▶ So that the program can be installed as disabled and then enabled when the Link to Liberty application is installed. This configuration can help to manage the availability of Link to Liberty programs during JVM server start and shutdown.

For the LinkToLiberty.java sample, we can define a PROGRAM with the attributes that are listed in Table 3-4.

Table 3-4 Attributes for a manually defined GETSUPPI PROGRAM definition

Attribute	Value
PROGRAM	GETSUPPI
Jvm	YES
Jvmclass	wlp:com.ibm.cicsdev.restappext.LinkToLiberty#getSupplierInfo
Jvmserver	ITSOJVM1
CONCURRENCY	REQUIRED
EXECKEY	CICS
Executionset	Dplsunset

All of the following prerequisites must be met for a manual Link to Liberty PROGRAM definition to be valid such that an EXEC CICS LINK call succeeds when the PROGRAM definition and the Link to Liberty method are installed and enabled:

- ▶ The program name matches @CICSProgram annotation.
- ▶ JVM is YES.
- ▶ Jvmclass matches the service name of the Link to Liberty method.
- ▶ Jvmserver matches the name of the Liberty JVM server.

If a PROGRAM definition with the same name as the @CICSProgram annotation is installed when the web application that contains the Link to Liberty method is installed in Liberty, no dynamic creation occurs. The only message that is issued is the DFHSJ1204 linkable service registration message to MSGUSR and *not* the associated DFHPG0101 resource definition message.

The manual definition of Link to Liberty programs can result in errors from a mismatch between the definition and the Link to Liberty linkable service. To reduce the likelihood of such errors, one approach is to allow dynamic program creation for the Link to Liberty methods on a test system. Then, the resulting created programs are copied when manual definitions for RDO are defined.

Note: Use Link to Liberty dynamic program creation on a test system to generate the correct program attributes. Then, copy these attributes to manual PROGRAM definitions. This process can be done in CICS Explorer or by using CEMT and CEDA/DFHCSDUP.

A dynamically created Link to Liberty PROGRAM definition is overwritten if a manual definition for the same program name is installed later.

3.2.6 Updating a Link to Liberty program

The method to update the target Java program for Link to Liberty depends on the original deployment method and configuration that was used. For more information, see Chapter 2, “Deploying a web application” on page 27.

To update a Link to Liberty program that is deployed as part of a CICS bundle, the updated version is exported to zFS and the CICS BUNDLE definition disabled and reenabled.

Note: A Link to Liberty application cannot be updated without an availability interruption with a single CICS Liberty server, whichever deployment method is used. The program is unavailable and link commands to the program fail.

In a multi-region configuration that uses IBM CICSplex® SM, dynamic program routing can be used to alleviate the availability interruptions that are caused during Link to Liberty application update. If an INVREQ or PGMIDERR response is received on a dynamically routed link request, CICSplex SM retries the request on another configured target region. As a result, an update can be rolled out across a set of Liberty-owning regions without an overall availability interruption.

If a Link to Liberty program uses a manual PROGRAM definition, it can be disabled while the Liberty application is updated, and then reenabled. The manual definition does not need to be changed unless the signature of the Java method for the program changed.

3.3 Qualities of service

In this section, we describe some other important aspects of Link to Liberty. This section features sample applications that are included in the `restapp-ext` project.

3.3.1 Transactions

A Link to Liberty Java program runs under the same CICS task and hence the same unit of work as the calling program. As a result, recoverable updates to resources from the calling program and the linked Java application are committed or backed out as a single unit of work.

However, Java Transaction API (JTA) with CICS integration within a linked Java application cannot be used. You can use JTA with integration turned off (for example, to commit JDBC updates performed by using a JDBC driver type 4 connection), but the JTA UserTransaction commits or rollbacks independently of the CICS unit of work.

To turn off JTA integration for a Liberty JVM server if `autoconfigure=true`, add the following option to the JVM profile and restart the server:

```
-Dcom.ibm.cics.jvmserver.wlp.jta.integration=false
```

Alternatively, add the following `cicsts_jta` element in `server.xml` and restart the server:

```
<cicsts_jta integration="false"/>
```

Note: JTA integration in a CICS Liberty JVM applies to the entire server and all applications that are running in it.

Because the linked Java application is restricted to the DPL subset of the CICS API, you cannot issue `Task.commit()` or `Task.rollback()` to syncpoint the unit of work while it is active in the JVM server.

In addition, a DPL to a Java Link to Liberty application fails with response `INVREQ` if `SYNCONRETURN` is specified because the Java application is unable to syncpoint.

Note: APAR PI83589 must be applied to receive the `INVREQ` response. Before this application, a syncpoint request is ignored.

DB2 transaction considerations

When JDBC is used, it is a common practice to use the `connection.commit()` API to commit a set of related updates to the database as a single unit of work. This configuration is not permitted in a linked Java application that uses a DB2 type 2 driver connection because it involves a CICS syncpoint.

However, `connection.commit()` is allowed for a type 4 driver connection. In this case, it is disconnected from the CICS unit of work and does not cause a syncpoint. This configuration is analogous to the case for JTA transactions with integration turned off.

Transactions summary

The allowed and forbidden transaction controls when Link to Liberty is used are listed in Table 3-5.

Table 3-5 Permitted transaction control APIs for Link to Liberty caller and target

Transaction control	Caller	Target (Java program)
SYNCPOINT or Task.commit()	Yes	No
SYNCPOINT ROLLBACK or Task.rollback()	Yes	No
DB2 type 2 Connection.commit() or rollback()	Not applicable	No
DB2 type 4 Connection.commit() or rollback()	Not applicable	Yes
User transaction with CICS JTA integration	Not applicable	No
User transaction without CICS JTA integration	Not applicable	Yes
CICS LINK with SYNCONRETURN	No	Yes (though not to another Link to Liberty program)

The rules regarding transactions in this section can be summarized as that you cannot do anything in a Link to Liberty Java program that causes the CICS task to attempt to perform a syncpoint (commit or rollback) explicitly or implicitly.

If you must perform a syncpoint, you must find an alternative way to accomplish this task. The following options are available:

- ▶ Pass back a response flag in a container to the caller of a Link to Liberty method to indicate whether the transaction should be committed or rolled back. The Link to Liberty caller can then issue a syncpoint on behalf of the linked program.
- ▶ Use JTA transactions with CICS integration turned off if CICS resources are not updated as part of the transaction.
- ▶ Use a DB2 type 4 driver connection instead of type 2 if only the DB2 updates must be committed together, and use the `connection.commit()` API.

The best option depends on the application and the resources that it updates.

3.3.2 Exception and abend processing

Exceptions are a part of the Java language. The Java application that is started by Link to Liberty can throw and catch Exceptions as with any other Java code.

Exceptions that are thrown and caught within the Link to Liberty Java program are internal to the program and have no external effect. For example, if you catch an Exception and handle it, an error code can be returned to the calling program in a response container.

The JCICS API typically throws a `CicsConditionException` in situations where the equivalent CICS API returns a response other than NORMAL or raise a condition. `CicsConditionException` can be caught and handled as with other Java Exceptions in which case no further action is taken and it becomes part of normal execution.

However, if a `CicsConditionException` is not caught in the Link to Liberty code and percolates back up the stack, it is caught by CICS and converted to a task abend as with an unhandled condition on the CICS API.

If an abend occurs or an Exception is thrown in a Link to Liberty method, you can choose to allow the Exception to percolate up the stack. This configuration causes the task to end with an AJ05 abend, or abend the task programmatically. A task abend ends the task and stops any further processing, which causes the unit of work to be rolled back.

To show the behavior of Link to Liberty with the different types of Abend and Exception that can occur (including transaction rollback), we used another sample Link to Liberty program that is available in the GitHub `restapp-ext` repository. The source for this `performAction()` method is in `LinkToTransaction.java` and is shown in Example 3-11.

Example 3-11 LinkToTransaction.java

```
private static final String CICSprog = "L2LTRAN";
@CICSProgram(CICSprog)
public void performAction() throws CicsConditionException,
UnsupportedEncodingException
{
    TSQ tsq = new TSQ();                                1
    tsq.setName(CICSprog);
    String itemStr = "Written from "+CICSprog+" by Task" +
        Task.getTask().getTaskNumber();
    int item = tsq.writeItem(itemStr.getBytes("IBM037"));
    Channel ch = Task.getTask().getCurrentChannel();      2
    Container actionCont = ch.getContainer("ACTION");
    String action = actionCont.getString().trim();

    switch(action) {                                     3
    case "":
        break;
    case "ABEND":
        Task.getTask().abend("AL2L");
        break;
    case "ROLLBACK":
        Task.getTask().rollback();
        break;
    case "COMMIT":
        Task.getTask().commit();
        break;
    case "THROW":
        throw new NullPointerException();
    case "CATCH":
    case "PERCOLATE":
        try {
            ItemHolder holder = new ItemHolder();
            tsq.readItem(item + 1, holder);
        }
        catch(CicsConditionException cce) {
            if (cce instanceof InvalidQueueIdException) {
            }
            if (action.equals("PERCOLATE")) {
                throw cce;
            }
        }
        break;
    default:
```

```

        throw new IllegalArgumentException("Invalid action: "
+action);
    }
}

```

Note: This example Link to Liberty program is provided to demonstrate the behavior of Exception handling in different situations. It is *not* intended as an example of a best practice.

In this example, the String parameter to the @CICSProgram annotation is defined as a static final variable, which is a permitted alternative to the use of a String literal. The Link to Liberty method performAction() that corresponds to PROGRAM L2LTRAN performs the following main steps as shown in Example 3-11 on page 84:

1. Writes an item to the temporary storage queue L2LTRAN (including the CICS task number) to demonstrate unit of work integration with the caller.
2. Gets the action String from the ACTION input container and trims it.
3. Runs a Java switch statement with the action as the expression. Depending on the action String, the corresponding case performs a transaction or Exception-related action.

The COBOL CICS front-end terminal program to start this Link to Liberty program is also in GitHub as /src/Cobo1/LINK2TXN.cb1. When used as a terminal transaction, it accepts an optional action as one of the following input parameters on the command line:

- ▶ COMMIT
- ▶ ROLLBACK
- ▶ ABEND
- ▶ THROW
- ▶ CATCH
- ▶ PERCOLATE

Program LINK2TXN performs the following steps:

1. Reads the action string from the terminal command line and converts to uppercase.
2. Writes an item to the temporary storage queue L2LTRAN (including the CICS task number) to demonstrate unit of work integration with the target.
3. Links to the Liberty program L2LTRAN that is passing the action command in the ACTION container.
4. If an error occurs on the link to L2LTRAN, rolls back the unit of work and issues an error message; otherwise, issues a completion message and returns to CICS.

To start LINK2TXN and the Link to Liberty method performAction(), run transaction JL2T at a terminal, optionally including one of the available actions as a parameter on the command line. The results of running this test case for each action are listed in Table 3-6.

Table 3-6 Action performed by performAction() method for each JL2T input

Action input	performAction()	Result
empty	No-op	Normal completion, updates to TSQ are committed.
COMMIT	Task.getTask().commit()	Task abend AEIP (INVREQ condition)
ROLLBACK	Task.getTask().rollback()	Task abend AEIP (INVREQ condition)
ABEND	Task.getTask().abend("AL2L")	Task abend AL2L

THROW	throw NullPointerException()	Task abend AJ05
CATCH	CicsConditionException (ITEMERR) caught within Java method.	Normal completion, updates to TSQ are committed.
PERCOLATE	CicsConditionException (ITEMERR) uncaught, percolates back to CICS.	Task abend AEIZ (ITEMERR condition not handled)
Anything else	IllegalArgumentException()	Task abend AJ05

To check the transaction integration between the COBOL caller and the linked Java program, examine the contents of the temporary storage queue L2LTRAN after running the JL2T transaction. An example is shown in Figure 3-19.

```

CEBR TSQ L2LTRAN          SYSID SC81 REC      1 OF      14      COL      1 OF      66
ENTER COMMAND ===>
***** TOP OF QUEUE *****
00001 WRITTEN FROM: LINK2TXN BY TASK: 00001939 FOR ACTION:
00002 Written from L2LTRAN by Task 1939
00003 WRITTEN FROM: LINK2TXN BY TASK: 00001942 FOR ACTION:      CATCH
00004 Written from L2LTRAN by Task 1942
00005 WRITTEN FROM: LINK2TXN BY TASK: 00001944 FOR ACTION:
00006 Written from L2LTRAN by Task 1944
00007 WRITTEN FROM: LINK2TXN BY TASK: 00003465 FOR ACTION:
00008 Written from L2LTRAN by Task 3465
00009 WRITTEN FROM: LINK2TXN BY TASK: 00003877 FOR ACTION:
00010 Written from L2LTRAN by Task 3877
00011 WRITTEN FROM: LINK2TXN BY TASK: 00004487 FOR ACTION:
00012 Written from L2LTRAN by Task 4487
00013 WRITTEN FROM: LINK2TXN BY TASK: 00004593 FOR ACTION:
00014 Written from L2LTRAN by Task 4593
***** BOTTOM OF QUEUE *****

PF1 : HELP          PF2 : SWITCH HEX/CHAR      PF3 : TERMINATE BROWSE
PF4 : VIEW TOP      PF5 : VIEW BOTTOM          PF6 : REPEAT LAST FIND
PF7 : SCROLL BACK HALF PF8 : SCROLL FORWARD HALF PF9 : UNDEFINED
PF10: SCROLL BACK FULL PF11: SCROLL FORWARD FULL PF12: UNDEFINED

```

Figure 3-19 CEBR view of the L2LTRAN Temporary Storage queue

When the transaction completes successfully and commits, two items are added to the queue, each including the same CICS task number. In all other cases, any added items are backed out during the transaction rollback and no new items appear.

As shown in Figure 3-19, entries are seen for action "" (none) or CATCH only because these actions are the only actions that result in the unit of work being committed. All other actions are backed out.

3.3.3 Security

When you link to a Java EE application from a CICS program, the user ID of the CICS task is passed into the Java EE application. It is automatically associated with any CICS work, such as JCICS calls that are made by the application.

The user ID of the CICS task is passed into the Liberty Java application and used to create the Java Subject. Liberty does not authenticate the user ID, although it does check that the user ID is present in the configured Liberty registry.

Where possible, use the SAF registry in Liberty because it checks the user ID that is passed in from CICS. If another type of user registry is used other than the SAF registry, and the CICS task user ID matches an ID in the non-SAF registry, that user ID is passed to the Java EE application.

If the same user ID is not present in the non-SAF registry, the Java EE application is linked with the unauthenticated user ID in the Java Subject, although the CICS task user ID is still used when any JCICS calls are made. The default unauthenticated user ID is WSGUEST, unless overridden with an unauthenticatedUser value in server.xml.

To configure security when linking a Java EE application, include the `cicsts:security-1.0` feature in your `server.xml` (see Example 3-12). If you do not include this feature, the user ID from the CICS task is not used to create the Java Subject, which is null. As a result, any authorization checks in your Java EE applications are invalid.

However, access to any CICS resources that use the JCICS API still run under the user ID of the CICS task.

Example 3-12 Adding the CICS security feature to server.xml

```
<feature>cicsts:security-1.0</feature>
```

Note: If `-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true` is in the JVM profile for the Liberty JVM server and `SEC=YES` is specified in the SIT, CICS adds `<feature>cicsts:security-1.0</feature>` and `<feature>ssl-1.0</feature>` to the `featureManager` section of `server.xml` when it starts the JVM.

By using another sample Link to Liberty method, we demonstrated the propagation of the CICS user ID that is associated with the calling task to the Java task.

The new sample is `LinkToSecurity.java` with its Link to Liberty method that is defined, as shown in Example 3-13.

Example 3-13 Java source for `getSecurityInfo()` method in `LinkToSecurity.java`

```
@CICSProgram("L2LSEC")
public void getSecurityInfo() throws CicsConditionException
{
    java.security.AccessControlContext context =
        java.security.AccessController.getContext();

    Subject activeSubject = Subject.getSubject(context);

    String principalName;

    if (activeSubject != null) {

        Set<Principal> principalSet = activeSubject.getPrincipals();

        Iterator<Principal> principalIterator = principalSet.iterator();
        Principal principal = principalIterator.next();

        if (principal != null) { // Get name from Principal.
            principalName = principal.getName();
        }
        else { // No Principal in Subject.
            principalName = "UNKNOWN";
        }
    }
    else {
        principalName = "NOSUBJECT";
    }
}
```

```

    }

    Channel ch = Task.getTask().getCurrentChannel();

    Container contPrincipal = ch.createContainer("PRINCIPAL");
    contPrincipal.putString(principalName);
    Container contUserid = ch.createContainer("USERID");
    String cicsUserid = Task.getTask().getUserID();
    contUserid.putString(cicsUserid);
}

```

The `getSecurityInfo()` method returns a response container that includes the CICS USERID from the task and the name of the Principal from the Java Subject, if it exists.

The sample COBOL program LINK2SEC in the `/src/Cobol` folder provides a terminal front end to this method and displays the results, as shown in the following example:

```
CICS USERID: JPLAW    Java Principal: CICSUSER
```

The results of running this sample with different combinations of CICS security (SEC=YES/NO) and the CICS Liberty security feature installed are listed in Table 3-7.

Table 3-7 CICS USERID and Principal Name used for a Link to Liberty call

CICS Security SEC=	Security feature installed	Principal name	CICS USERID
NO	NO	No Java Subject	CICSUSER (default user)
NO	YES	No Java Subject	CICSUSER (default user)
YES	NO	No Java Subject	JPLAW (caller user)
YES	YES	JPLAW (caller user)	JPLAW (caller user)

The Java Subject is available in only the Context if CICS security is enabled and `<feature>cicsts:security-1.0</feature>` is installed in the Liberty server.

Note: If a `safCredentials` element is in `server.xml`, such as `<safCredentials profilePrefix="SC8CICS" unauthenticatedUser="CICSUSER"/>`, the CICS region user ID and the Link to Liberty caller user ID must have READ access to the specified `profilePrefix` profile in RACF APPL class. Otherwise, the RACF user ID check fails and the specified `unauthenticatedUser` (default WSGUEST) is used as the Principal name.

3.3.4 Summary

In this chapter, we described the new Link to Liberty feature in CICS TS. This feature allows a Java method in an installed Liberty web application to be started as a CICS linkable program that passes data in CICS containers by using a channel.

After an introduction to the concepts of the new capability, we described the development, deployment, and runtime uses with the `restapp-ext` sample application. This application is available to download from the `cicsdev` GitHub repository.

Finally, we described some important quality of service aspects of Link to Liberty. We concentrated on transactions, exception, and abend handling, and security. These aspects were demonstrated with more sample Link to Liberty applications, which also are available in the GitHub repository.



Connecting to Db2 by using JDBC

In this chapter, we describe how to use Java database connectivity (JDBC) within a Java EE application in CICS Liberty to access data in a Db2 database. Our scenario uses Db2 v12 for z/OS with CICS TS V5.4.

We also show how to configure a CICS Liberty JVM server to use JDBC type 2 and type 4 connectivity. Finally, we describe how to integrate Db2 updates with other CICS transactional updates.

This chapter includes the following topics:

- ▶ 4.1, “JDBC overview” on page 92
- ▶ 4.2, “Installing the JDBC Employee application” on page 94
- ▶ 4.3, “Using JDBC type 2 connectivity” on page 97
- ▶ 4.4, “Using JDBC type 4 connectivity” on page 104
- ▶ 4.5, “Transactional support with JDBC” on page 108
- ▶ 4.6, “Tracing JDBC” on page 110

4.1 JDBC overview

JDBC is the Java specification of a standard application programming interface (API) that allows Java programs to access database management systems. The JDBC API consists of a set of interfaces and classes that are written in the Java programming language. The JDBC API is divided into two packages: `java.sql` and `javax.sql`, which are included in the Java SE and Java EE platforms.

By using these standard interfaces and classes, programmers can write applications that connect to databases, send queries that are written in structured query language (SQL), and process the results.

Three recent versions of the JDBC 4 specification JDBC 4.0, JDBC 4.1, and JDBC 4.2 are available. JDBC 4.0 is supported in CICS Liberty by using the `jdbbc-4.0` feature; JDBC 4.1 is supported by using the `jdbbc-4.1` feature.

4.1.1 JDBC drivers

Because JDBC is a standard specification, one Java program that uses the JDBC API can connect to any database management system (DBMS) if a driver exists for that particular DBMS.

The following commonly used types of JDBC drivers are available for use in CICS Liberty JVM servers:

- ▶ Type 2: These drivers use the client libraries of a locally installed database and convert the JDBC method calls into the native calls of the local database driver.
- ▶ Type 4: These drivers are pure Java drivers that connect directly to a remote database server over the network.

Db2 for z/OS provides JDBC support through the IBM Data Server Driver for JDBC and SQLJ, which is known as the *JCC driver*. This driver is a unified driver that combines the type 2 and type 4 connectivity implementations in one driver. The following versions of the JCC driver are available with Db2 for z/OS:

- ▶ `db2jcc.jar` for JDBC 3.0 and earlier support
- ▶ `db2jcc4.jar` for JDBC 3.x and 4.x support

In this chapter, we used the `db2jcc4.jar` driver.

JDBC connectivity

Although the type 2 and type 4 modes of JDBC connectivity are different, both are supported in the CICS Liberty JVM server environment for connecting to IBM Db2 for z/OS. Which connectivity type is best suited to a specific CICS Liberty application depends on the factors that are listed in Table 4-1.

Table 4-1 Comparison of type2 and type4 JDBC drivers

Qualities of service	JDBC type 2 connectivity	JDBC type 4 connectivity
Connectivity	Must connect to local Db2 database by using the CICS DB2 attachment.	Can connect to any database by using a TCP/IP network.
Performance	Shorter path length offers better response times. ^a	zIIP offload in the Java driver and in the Db2 DDF can be used to reduce CPU costs.
Security	Security context that is inherited from the CICS transaction.	Credentials are configured in the data source definition.
Transactionality	Integrated with CICS unit-of-work and CICS sync point control.	Requires usage of Connection.commit() or Java Transaction API (JTA) to coordinate updates.

a. For more information about the performance of JDBC type 2 and type 4 connectivity in CICS Liberty, see the IBM Redpaper™ publication [IBM CICS Performance Series: Comparing Type 2 and Type 4 JDBC Driver Performance with IBM CICS Transaction Server for z/OS V5.2 Liberty JVM server](#).

4.1.2 Data sources

Before you can run SQL statements in any SQL program, you must be connected to a database. In the Java platform, the connection set up to a database from a server is known as a *data source*.

A Java application can establish a connection to a data source by using the JDBC DriverManager or DataSource interfaces, which are part of the `java.sql` package. The use of the DriverManager interface limits portability and the DataSource is the standard model that is used for most Java EE applications. The use of a DataSource allows the application code to remain version independent of the database connection details, such as the JDBC driver connection type.

When a Java EE application in Liberty connects to a data source by using the DataSource interface, the connection details are typically defined in the Liberty server configuration file. Each application can then refer to this data source by using a logical name, which is located by using the Java Naming and Directory Interface (JNDI).

The DataSource object is obtained by using the `JNDI InitialContext.doLookup()` method or the resource injection. Having obtained a DataSource instance, the `DataSource.getConnection()` method is then used to obtain the database connection.

4.1.3 Static and dynamic SQL

JDBC can use dynamic or static SQL. Static SQL uses predefined SQL operations that do not change when the program is run. Dynamic SQL operations are not predefined and the underlying database operations can change when the program is run. Within Java, the use of static SQL requires the use of the SQLJ API, which is a set of programming extensions that allow Java programs to embed SQL statements.

SQLJ requires another compilation step to convert the `.sqlj` source files into Java code before they are compiled by the Java compiler. The dynamic and static models feature the following key characteristics:

- ▶ **Dynamic SQL**

Because it does not require any special preparation process, Dynamic SQL is easier to develop and deploy than SQLJ.

It also can be written by using standard JDBC calls or by using the Java Persistence Architecture (JPA). JPA simplifies programming at a cost of less control over database events and an increased number of smaller database interactions.

- ▶ **Static SQL**

This model can be more efficient because the database knows what should be done. It also can be more reliable because variable type errors are detected at compilation time.

SQLJ code is compact and partially generated by tools, such as IBM Data Studio. SQLJ also supports more granular security options. The use of prepared statements can help avoid SQL injection attacks.

In this chapter, we use dynamic SQL and the JDBC API, which provides the simplest route to get started with Java access to Db2 data.

4.2 Installing the JDBC Employee application

In this IBM Redbooks publication, we provide a sample JDBC application that is named the JDBC Employee application. This application queries employee information from the sample Db2 EMP table and supports creating, reading, updating, and deleting actions to be performed on the entries in the database.

The source for our application is available for download at [the CICSDev GitHub repository](#). The instructions for deploying the Employee application are provided in the `.readme` file in the GitHub repository.

The Employee application uses a data source configuration to define the connection to the underlying database. When Db2 for z/OS is used, this connection can be configured to use JDBC type 2 or type 4 connectivity. Sample instructions are provided for configuring and installing the application into CICS Liberty.

Whichever type of JDBC driver is used, you must configure the following resources:

- ▶ Liberty features
- ▶ Liberty `<dataSource>` element
- ▶ CICS resource definitions:
 - TSMODEL
 - URIMAP
 - TRANSACTION

If you use JDBC type 2 connectivity in CICS, you also must define the following resources:

- ▶ DB2CONN
- ▶ DB2ENTRY
- ▶ DB2TRAN

Instructions for configuring these resources are provided in the following sections.

4.2.1 Liberty features

In addition to the JDBC feature, the Employee application uses the JNDI to locate the DataSource and JavaServer Faces (JSF) technology to build the web pages.

We added the Java EE 7 versions of these Liberty features that are shown in Example 4-1 to the `<featureManager>` list element in our Liberty `server.xml` configuration file.

Example 4-1 Liberty features

```
<featureManager>
  <feature>jndi-1.0</feature>
  <feature>jsf-2.2</feature>
  <feature>jdbc-4.1</feature>
</featureManager>
```

The Employee web application can be defined into Liberty by using an application element or a CICS bundle project. We used the application element that is shown in Example 4-2 to deploy the Employee application.

Example 4-2 Liberty application element

```
<application
  location="/var/cicsts/SC8CICS2/ITS0JVM1/apps/ employee.jdbc.web.war">
</application>
```

4.2.2 Data source definition

In addition to the Liberty features and application definition, the data source must be defined in the Liberty server configuration file. This configuration is achieved by using a `<dataSource>` element and an associated `<library>` element in the `server.xml` file.

For more information about the configuration for the two different modes of JDBC connectivity (type 2 and type 4), see 4.3.2, “Configuring `server.xml`” on page 100 and 4.4.2, “Configuring `server.xml`” on page 105. Several `server.xml` samples are provided with the GitHub repository.

4.2.3 CICS resources

Our sample Employee application requires the following CICS resources definitions, which are required for type 2 and type 4 connectivity scenarios:

- ▶ TSMODEL
- ▶ URIMAP
- ▶ TRANSACTION

More CICS resource definitions, including DB2CONN, DB2TRAN, and DB2ENTRY definitions, are required to use type 2 connectivity. For more information, see 4.3.1, “Configuring CICS resources” on page 98.

TSMODEL

Our ITSO Employee application uses a CICS temporary storage queue (TSQ) that is named DB2LOG to log updates that are made to the Db2 database. This TSQ is defined as recoverable so that updates to it are coordinated as part of the CICS unit-of-work.

We defined a TSMODEL that is named DB2LOG, as shown in Figure 4-1. The Recovery attribute was set to Yes to define the TSQ as recoverable by CICS.

```

OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA ALTER TSmodel( DB2LOG )
  TSmodel      : DB2LOG
  Group        : DB2
  DEScription  ==> RECOVERABLE TS FOR DB2
  PRefix       ==> DB2LOG
  XPrefix      ==>
  Location     ==> Auxiliary      Auxiliary | Main
  EXPIRYINT    : 00000          0-15000 (Hours)
  EXPIRYINTMin ==> 0000000      0-900000 (Minutes)
  RECOVERY ATTRIBUTES
  REcovery     ==> Yes          No | Yes
  SECURITY ATTRIBUTES
  Security     ==> No          No | Yes
  SHARED ATTRIBUTES
  POolname     ==>
  REMOTE ATTRIBUTES
  REMOTESystem ==>
+  REMOTEPrefix ==>

                                           SYSID=SC82 APPLID=SC8CICS2

PF 1 HELP 2 COM 3 END          6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL

```

Figure 4-1 CICS TSMODEL definition

URIMAP and TRANSACTION

Next, we added a URIMAP and a TRANSACTION definition. These definitions enabled us to map the HTTP requests to our Employee application, such that they ran under a unique transaction ID. Using a URI map and transaction ID allows requests to this application to be easily secured and monitored. Also, when type 2 connectivity is used, the transactions can be mapped to the required DB2TRAN definition, which is used to specify the Db2 plan name.

The URI map definition EMPJDB2 is shown in Figure 4-2. It is used to map the URL path /employee.jdbc.web/* to our transaction ID JDB2. We allowed the Host and Port parameters to default so that these parameters were not part of the filter criteria.

```

OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA Alter Urimap( EMPJDB2 )
+ Port ==> No                                     No | 1-65535
  Host ==> *
  Path ==> employee.jdbc.web/*
  (Mixed Case) ==>
  ==>
  ==>
  OUTBOUND CONNECTION POOLING
  Socketclose ==>                                0-240000 (HHMMSS)
  ASSOCIATED CICS RESOURCES
  TCPIPService ==>
  Analyzer ==> No                                 No | Yes
  Converter ==>
  Transaction ==> JDB2
  Program ==>
+ Pipeline ==>

SYSID=SC82 APPLID=SC8CICS2
PF 1 HELP 2 COM 3 END          6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL

```

Figure 4-2 URIMAP definition

Next, we created and installed the TRANSACTION definition JDB2 (see Figure 4-3). This definition is a copy of the CJSA definition, which is the default JVM server request processor transaction.

```

OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA Alter TRANSAction( JDB2 )
  TRANSAction : JDB2
  Group : DB2
  DEScription ==> JVMSEVER request processor transaction for employee app
  PROGRAM ==> DFHSJTHP
  TWasize ==> 000000                                0-32767
  PROFile ==> DFHCICST
  PArtitionset ==>
  STatus ==> Enabled                                Enabled | Disabled
  PRIMedsize : 000000                                0-65520
  TASKDATAloc ==> Any                                Below | Any
  TASKDATAKey ==> User                                User | Cics
  STOrageclear ==> No                                No | Yes
  RUNaway ==> System                                System | 0 | 250-2700000
  SHUttdown ==> Disabled                            Disabled | Enabled
  ISolate ==> Yes                                    Yes | No
  Brexit ==>
+ REMOTE ATTRIBUTES

SYSID=SC82 APPLID=SC8CICS2
PF 1 HELP 2 COM 3 END          6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL

```

Figure 4-3 TRANSACTION definition JDB2

With these definitions, we installed them into our CICS region by using the **CEDA INSTALL GROUP(DB2)** command.

4.3 Using JDBC type 2 connectivity

In a CICS Liberty environment, the use of the Db2 JCC driver with type 2 connectivity converts the JDBC requests into their EXEC SQL equivalents. The converted requests flow into the CICS Db2 attachment facility in the same way as EXEC SQL requests from COBOL or other non-Java programs. The customization and tuning options for the CICS Db2 attachment facility apply equally to Java and non-Java programs.

4.3.1 Configuring CICS resources

To add support for JDBC type 2 connectivity, we changed our CICS region's configuration. This process is described next.

CICS STEPLIB

The first step in adding Db2 support to our CICS region is to make the Db2 libraries available to your CICS region by using the MVS LNKLIST or the region's STEPLIB. We added the SDSNLOAD and SDSNLOD libraries to the STEPLIB, as shown in Example 4-3.

Example 4-3 CICS region STEPLIB

```
//STEPLIB DD DSN=CICSTS54.CICS.SDFHAUTH,DISP=SHR
//        DD DSN=CICSTS54.CPSM.SEYUAUTH,DISP=SHR
//        DD DSN=CICSTS54.CICS.SDFJAUTH,DISP=SHR
//        DD DSN=CICSTS54.SDFHLIC,DISP=SHR
//        DD DSN=CEE.SCEECICS,DISP=SHR
//        DD DSN=DB2AT.SDSNLOAD,DISP=SHR
//        DD DSN=DB2AT.SDSNLOD2,DISP=SHR
```

CICS resource definitions

To install our Employee application by using a Db2 type 2 connection, we configured the following CICS resource definitions:

- ▶ DB2CONN
- ▶ DB2ENTRY
- ▶ DB2TRAN

These resources definitions are described in the following sections.

DB2CONN

Next, we added a CICS DB2CONN resource definition. This resource defines the connection from CICS to the Db2 subsystem (see Figure 4-4).

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA ALTER DB2Conn( D2AG )
  DB2Conn      : D2AG
  Group        : DB2
  Description ==>
CONNECTION ATTRIBUTES
CONnectorr ==> Sqlcode          Sqlcode | Abend
DB2Groupid ==> D2AG
DB2Id ==>
MSGQUEUE1 ==> CDB2
MSGQUEUE2 ==>
MSGQUEUE3 ==>
Nontermrel ==> Yes             Yes | No
PUrgecycle ==> 00 , 30         0-59
RESyncmember ==> Yes           Yes | No
REUselimit ==> 01000           0-10000
SIgnid ==>
STANdbymode ==> Reconnect      Reconnect | Connect | Noconnect
+ STATsqueue ==> CDB2

SYSID=SC82 APPLID=SC8CICS2

PF 1 HELP 2 COM 3 END                6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 4-4 CICS DB2CONN: Part I

The DB2Groupid attribute was set to the group ID of D2AG, which was the DB2 group ID of our Db2 data sharing group (see Figure 4-5).

```

OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA ALTER DB2Conn( D2AG      )
+ TCblimit      ==> 0012                        4-2000
  THREADError   ==> N906D                      N906D | N906 | Abend
POOL THREAD ATTRIBUTES
  ACcountrec    ==> None                       None | TXid | TAsk | Uow
  AUTHId        ==>
  AUTHType      ==> Userid                     Userid | Opid | Group | Sign | TTerm
                                           | TX
  DRollback     ==> Yes                       Yes | No
  PLAN          ==>
  PLANExitname  ==> DSNCEXT
  PRIority      ==> High                      High | Equal | Low
  THREADLimit   ==> 0003                      3-2000
  THREADWait    ==> Yes                      Yes | No
COMMAND THREAD ATTRIBUTES
  COMAUTHId     ==>
  COMAUTHType   ==> Userid                     Userid | Opid | Group | Sign | TTerm
                                           | TX
+
                                           SYSID=SC82 APPLID=SC8CICS2

PF 1 HELP 2 COM 3 END                        6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL

```

Figure 4-5 CICS DB2CONN: Part II

The AUTHType attributes and the COMAUTHType attributes were allowed to default to Userid, which signifies that authorization to the database uses the CICS task user ID.

DB2ENTRY and DB2TRAN

Next, we added a DB2ENTRY and DB2TRAN definition, as shown in Figure 4-6 and Figure 4-7 on page 100.

```

OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA ALTER DB2Entry( JDB2      )
  DB2Entry      : JDB2
  Group         : DB2
  DEScription   ==> DB2 entry definition for JDBC type 2 connection █
THREAD SELECTION ATTRIBUTES
  TRansid       ==> JDB2
THREAD OPERATION ATTRIBUTES
  ACcountrec    ==> None                       None | TXid | TAsk | Uow
  AUTHId        ==>
  AUTHType      ==> Userid                     Userid | Opid | Group | Sign | TTerm
                                           | TX
  DRollback     ==> Yes                       Yes | No
  PLAN          ==> ITSQJCC
  PLANExitname  ==>
  PRIority      ==> High                      High | Equal | Low
  PROtectnum    ==> 0010                      0-2000
  THREADLimit   ==> 0010                      0-2000
+  THREADWait    ==> Pool                      Pool | Yes | No
                                           SYSID=SC82 APPLID=SC8CICS2

PF 1 HELP 2 COM 3 END                        6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL

```

Figure 4-6 CICS DB2ENTRY definition

```

OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA ALTER DB2Tran( JDB2      )
  DB2Tran      : JDB2
  Group        : DB2
  Description   ==> DB2TRAN definition for JDBC type 2
  Entry         ==> JDB2
  Transid      ==> JDB*
DEFINITION SIGNATURE
  DEFInetime    : 11/22/17 08:18:54
  CHANGETime    : 11/22/17 08:18:54
  CHANGEUsrid   : CICSUSER
  CHANGEAGEnt   : CSDApi          CSDApi | CSDBatch
  CHANGEABEl    : ..A710.

                                           SYSID=SC82 APPLID=SC8CICS2

PF 1 HELP 2 COM 3 END                6 CSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL

```

Figure 4-7 CICS DB2TRAN definition

4.3.2 Configuring server.xml

To configure the use of JDBC type 2 connectivity with CICS Liberty, the following resources must be added:

- ▶ The Liberty JSF, and JDBC features that are required by the application. For more information, see 4.2.1, “Liberty features” on page 95.
- ▶ A `<library>` element that refers to the JCC driver and license file.
- ▶ A `<dataSource>` element that defines the connection to the database by using the CICS Db2 attachment.

Our `server.xml` with this updated configuration is shown in Example 4-4.

Example 4-4 Liberty `server.xml`: Type 2 `dataSource`

```

<featureManager>
  <feature>jndi-1.0</feature>
  <feature>jsf-2.2</feature>
  <feature>jdbc-4.1</feature>
</featureManager>

<library id="Db2Lib">
  <fileset dir="/usr/lpp/db2c10/db2a/jdbc/classes"
    includes="db2jcc4.jar db2jcc_license_cisuz.jar" />
  <fileset dir="/usr/lpp/db2c10/db2a/jdbc/lib"
    includes="libdb2jcc2zos4_64.so" />
</library>

<dataSource id="db2type2"
  jndiName="jdbc/sample"
  transactional="false">
  <jdbcDriver libraryRef="jdbcLib" />
  <connectionManager agedTimeout="0" />
  <properties.db2.jcc driverType="2"
    currentSchema="DSN81210" />
</dataSource>

```

```
<application
  location="/var/cicsts/SC8CICS2/ITS0JVM1/apps/ employee.jdbc.web.war">
</application>
```

Note: In releases before CICS TS V5.4, the `<dataSource>` element was supported for type 4 connectivity to Db2 only. Instead, type 2 connectivity required the use of the CICS provided `<cicsts_datasource>` and `<cicsts_jdbcDriver>` elements.

The new type 2 `<dataSource>` support in CICS TS V5.4 provides a more standard implementation for JDBC support, and is supported in CICS TS V5.3 with APAR PI77502. If applications are migrated from the use of the `<cicsts_dataSource>` element to the `<dataSource>` element, the commit processing logic must be reviewed in your JDBC application because of changes in the default behavior for commit on cleanup. For more information, see 4.5, “Transactional support with JDBC” on page 108.

Data source definition

We specified the following attributes on our `<dataSource>` element:

- ▶ `jndiName="jdbc/sample"`

This attribute is the JNDI name that is used by our Employee application to locate the `DataSource` object. It is specified on the `@Resource` annotation on the `DataSource` field in the `DatabaseOperationsManager` class.

```
@Resource(authenticationType=AuthenticationType.CONTAINER,
          name="jdbc/sample")
private DataSource ds;
```

- ▶ `transactional="false"`

The `<transactional>` attribute must be set to `false` to ensure that the data source updates are not coordinated by the Liberty transaction manager. If this setting is left to default to `true`, Liberty uses RRS as the transaction coordinator when JDBC type 2 connectivity is used, which does not function correctly in the CICS environment.

Setting this value to `false` disables transaction coordination by Liberty and allows the updates to be coordinated as part of the CICS unit-of-work through the CICS Db2 attachment.

- ▶ `jdbcDriver libraryRef="jdbcclib"`

The `libraryRef` attribute on the `<jdbcDriver>` sub-element must reference the `<library>` element that is used to configure the location of the JCC driver `db2jcc4.jar` and the accompanying license file `db2jcc_license_cisuz.jar`.

- ▶ `connectionManager agedTimeout="0"`

The `<connectionManager>` element should specify `agedTimeout=0` to ensure that the Liberty data source connection pooling is disabled because the database connections are pooled instead by using the CICS Db2 attachment.

- ▶ `properties.db2.jcc`

Attributes that are supplied in the `<properties.db2.jcc>` child element are passed directly to the Db2 JCC driver. We specified the following values:

- `driverType`

The `driverType` attribute must specify 2 to signify the use of JDBC type 2 connectivity with the Db2 JCC driver.

- `currentSchema`

The currentSchema specifies the default schema name that is used to qualify unqualified database objects in dynamically prepared SQL statement. We used the value currentSchema="DSN81210" because Db2 V12 uses DSN81210 as the sample schema.

4.3.3 Binding the plan

Finally, we must bind the ITSOJCC plan with a PKLIST of NULLID.*. The ITSOJCC plan was specified as our plan name in our DB2ENTRY definition, as shown in Example 4-5.

Example 4-5 Db2 bind plan for JDBC

```
//D2A1JCCP JOB (999,P0K),'D2A1 INSTALL',CLASS=A,
// MSGCLASS=T,NOTIFY=&SYSUID,TIME=NOLIMIT,REGION=0M
/*JOBPARM SYSAFF=SC80,L=9999
// JCLLIB ORDER=(DB2AM.PROCLIB)
//JOBLIB DD DISP=SHR,DSN=DB2AT.SDSNLOAD
// DD DISP=SHR,DSN=CEE.SCEERUN
//PH01PS02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(D2AG)
BIND PLAN(ITSOJCC) OWNER(SYSADM)ACTION(REPLACE)PKLIST(NULLID.*)+
RETAIN CURRENTDATA(NO) ISO(CS) ENCODING(EBCDIC) SQLRULES(DB2)
RUN PROGRAM(DSNTEP2) PLAN(DSNTEP12) +
LIB('DB2AM.RUNLIB.LOAD') PARM('/ALIGN(MID)')
END
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT EXECUTE, BIND ON PLAN ITSOJCC TO PUBLIC;
/*
```

4.3.4 Running the application

The Employee web application is accessed on our CICS Liberty server by using the following URL:

`http://wtsc80:52080/employee.jdbc.web/`

This URL loads the Employee database list form, as shown in Figure 4-8.

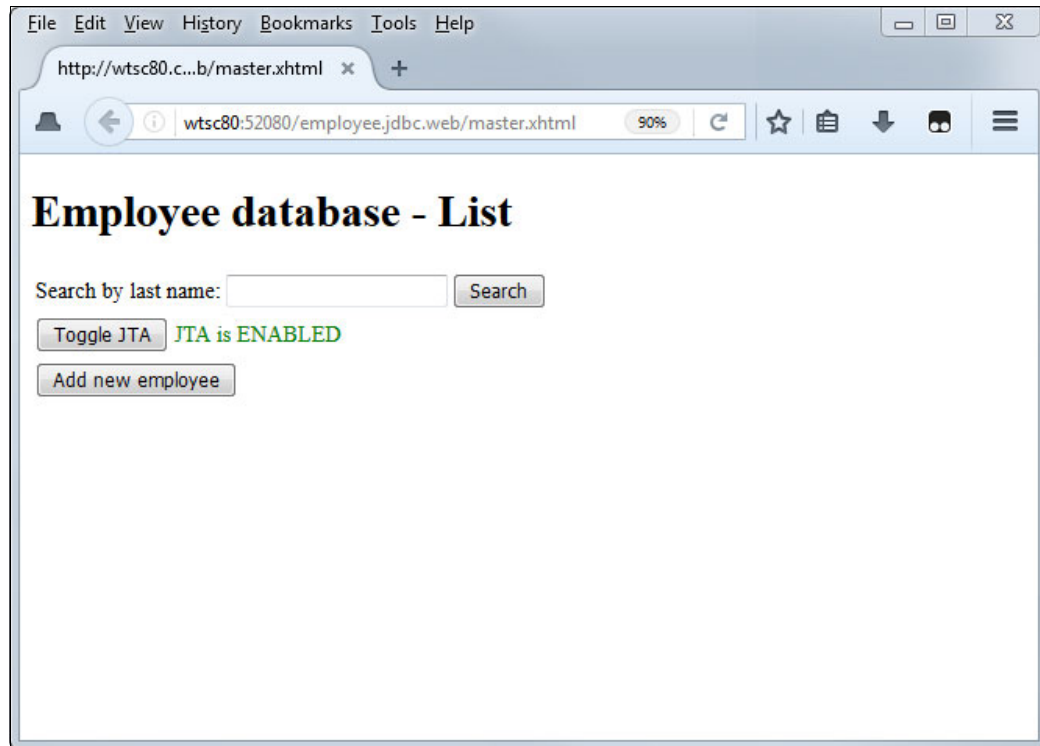


Figure 4-8 Employee database list form

From this form, you can disable and enable the use of Java Transaction API (JTA) for commit processing by selecting **Toggle JTA**. For more information about how to use JTA, see 4.5, “Transactional support with JDBC” on page 108.

Next, you add entries or query entries in the EMP table. If you leave the search field blank and click **Search**, all of the employees from the table are returned, as shown in Figure 4-9 on page 104.

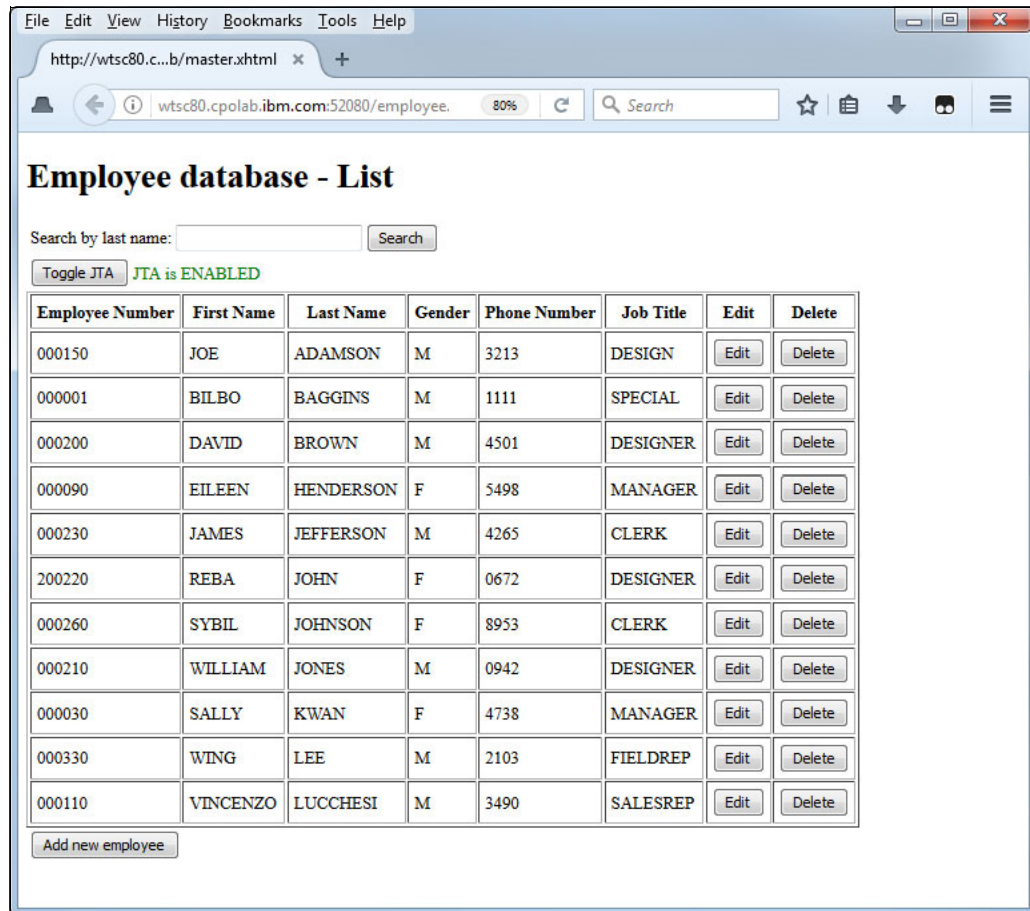


Figure 4-9 Employee application search results

If the employee information is returned, this result confirms that a connection was made to the DB2 database and that the EMP table was read successfully.

If any errors are returned by the application, further diagnostics often can be found in the JVM server stderr destination.

4.4 Using JDBC type 4 connectivity

The Liberty JVM server also supports type 4 connectivity. This Type 4 connectivity uses TCP/IP to connect to a remote Db2 subsystem through the support of the Db2 Distributed Data Facility (DDF) instead of the use of the CICS Db2 attachment and associated DB2CONN resource. The DDF is a built-in component of Db2 and provides the network connectivity to and from other servers or clients.

4.4.1 Configuring CICS resources

When JDBC type 4 connectivity is used, it is not necessary to define any specific CICS resource definitions. However, we used the same CICS TRANSACTION, URIMAP, and TSMODEL resource definitions for our JDBC type 4 connectivity configuration as for our type 2 configuration. For more information about how to configure these resources, see 4.2.3, “CICS resources” on page 95.

4.4.2 Configuring server.xml

To configure the use of JDBC type 4 connectivity with CICS Liberty, the following resources must be added:

- ▶ The Liberty JSF, JNDI, and JDBC features that are required by the application. For more information, see 4.2.1, “Liberty features” on page 95.
- ▶ A `<library>` element that refers to the JCC driver and license file.
- ▶ A `<dataSource>` element that defines the connection to the database.

Our `server.xml` with this updated configuration is shown in Example 4-6.

Example 4-6 Liberty server.xml: Type 4 data source

```
<featureManager>
  <feature>jndi-1.0</feature>
  <feature>jsf-2.2</feature>
  <feature>jdbc-4.1</feature>
</featureManager>
<library id="Db2Lib">
  <fileset dir="/usr/lpp/db2c10/db2a/jdbc/classes"
    includes="db2jcc4.jar db2jcc_license_cisuz.jar" />
  <fileset dir="/usr/lpp/db2c10/db2a/jdbc/lib"
    includes="libdb2jcct2zos4_64.so" />
</library>

<dataSource id="db2type4"
  jndiName="jdbc/sample"
  type="javax.sql.XADataSource">
  <jdbcDriver libraryRef="jdbcLib" />
  <connectionManager maxPoolSize="50" />
  <properties.db2.jcc driverType="4"
    databaseName="DB2A"
    currentSchema="DSN81210"
    serverName="localhost"
    portNumber="38000"
    user="DB2USER"
    password="{xor}0z1tLz4sLA==" />
</dataSource>
<application
  location="/var/cicsts/SC8CICS2/ITS0JVM1/apps/ employee.jdbc.web.war">
</application>
```

The `<dataSource>` element specifies the following XML attributes:

- ▶ `jndiName="jdbc/sample"`

This attribute is the JNDI name that is used by our Employee application to locate the `DataSource` object. It is specified on the `@Resource` annotation on the `DataSource` field in the `DatabaseOperationsManager` class, as shown in the following example:

```
@Resource(authenticationType=AuthenticationType.CONTAINER,
  name="jdbc/sample")
private DataSource ds;
```

- ▶ `type="javax.sql.XADataSource"`

The `type` attribute specifies the type of `DataSource` that is provided by the JDBC driver. The following options are available:

- `javax.sql.DataSource`
- `javax.sql.ConnectionPoolDataSource`
- `javax.sql.XADataSource`

We specified `javax.sql.XADataSource` to enable our `DataSource` to support the XA two-phase commit protocol, which allows updates to Db2 to be coordinated as part of a JTA transaction. For more information about transaction support, see 4.5, “Transactional support with JDBC” on page 108.

- ▶ `transactional="true"`

The `transaction` element should be allowed to default to `true` so that the database updates are managed by the Liberty transaction manager. We did not specify this attribute because the default value is `true`.

- ▶ `jdbcDriver libraryRef="jdbcclib"`

The `<jdbcDriver>` `libraryRef` attribute must reference the `<library>` element that is used to configure the location of the JCC driver `db2jcc4.jar` and the accompanying license file `db2jcc_license_cisuz.jar`.

- ▶ `<connectionManager maxPoolSize />`

The `<connectionManager>` element can be used to control attributes of the connection pool. We set the pool size to 50 to equal the number of configured JVM server threads.

- ▶ `properties.db2.jcc`

The following attributes can be used:

- `driverType`

The `driverType` attribute must be set to 4 to specify the use of JDBC type 4 connectivity with the Db2 JCC driver.

- `databaseName`

This attribute is the Db2 location value. On our Db2 subsystem, it was set to the value `DB2A`.

- `currentSchema`

This attribute specifies the default schema name that is used to qualify unqualified database objects in dynamically prepared SQL statement. We used the value `currentSchema="DSN81210"` because Db2 V12 uses `DSN81210` as the sample schema.

- `serverName`

This attribute is the TCP/IP host on which the Db2 DDF facility is configured to listen. We used the value `localhost` because we were connecting to a local Db2 on the same MVS image.

- `portNumber`

This attribute is the port on which the Db2 DDF facility is configured to listen. Our Db2 DDF was configured to use port 38000.

The location, server name, and port in use by the DDF can be discovered by using the DB2 command **-DISPLAY DDF** from the DB2I primary option menu in SDSF, as shown in Example 4-7.

Example 4-7 DISPLAY DDF command output

```
DSNL080I  -D2A1 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
```

```

DSNL081I STATUS=STARTD
DSNL082I LOCATION          LUNAME          GENERICCLU
DSNL083I DB2A              USIBMSC.SCPD2A1  -NONE
DSNL084I TCPPORT=38000 SECPORT=38001 RESPORT=38002 IPNAME=-NONE
DSNL085I IPADDR=::9.76.61.131
DSNL086I SQL      DOMAIN=wtsc80.cpolab.ibm.com
DSNL086I RESYNC  DOMAIN=wtsc80.cpolab.ibm.com
DSNL089I MEMBER IPADDR=::9.76.61.131
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL106I SESSIDLE = 001440
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE

```

- user and password

These attributes supply the user ID and password credentials that are used for authenticating with Db2. The password can be XOR or AES encrypted by using the Liberty `securityUtility` script. This utility can be started by using the CICS-generated `wlpenv` script, which is in the JVM server working directory, as shown in Example 4-8.

Example 4-8 Use of `wlpenv` to run `securityUtility`

```

$ [SC80] /cicsts/SC8CICS2/ITS0JVM1: ./wlpenv securityUtility encode <password>
Executing: /usr/lpp/cicsts/cicsts54/wlp/bin/securityUtility encode <password>
JAVA_HOME=/usr/lpp/java/J8.0_64_SR4
WLP_INSTALL_DIR=/usr/lpp/cicsts/cicsts54/wlp
WLP_USER_DIR=/var/cicsts/SC8CICS2/wlp
WLP_OUTPUT_DIR=/var/cicsts/SC8CICS2/wlp/servers
SERVER_NAME=itsowlp1
{xor}0z1tLz4sLA==

```

4.4.3 Running the application

The Employee web application is accessed on our CICS Liberty server by using the following URL:

`http://wtsc80:52080/employee.jdbc.web/`

This URL displays the Employee database list form and should function as before when JDBC type 2 connectivity is used, as described in 4.3.4, “Running the application” on page 103.

4.4.4 Container managed security

The application server can be configured to use container managed authentication aliases to provide the user ID and password credentials for JDBC type 4 data sources. This configuration allows the credentials to be separated from the configuration of the data source and shared between data sources.

To enable this method of authentication, the application must use a resource injection annotation for the JNDI lookup of the `DataSource` rather than a direct JNDI lookup.

A `DataSource` resource injection is defined in our application by using the field level annotation in the `DatabaseOperationsManager` class that is shown in Example 4-9.

Example 4-9 Resource annotation for DataSource

```
@Resource(authenticationType=AuthenticationType.CONTAINER,  
    name="jdbc/sample")  
private DataSource ds;
```

To enable the use of the authentication alias, we added an <authData> element to our Liberty server.xml. Then, we updated the <dataSource> element with a containerAuthDataRef attribute that identifies the <authData> element (see Example 4-10).

Example 4-10 server.xml updates for container managed security

```
<dataSource id="db2type4"  
    jndiName="jdbc/sample"  
    type="javax.sql.XADataSource"  
    containerAuthDataRef="db2user">  
    <jdbcDriver libraryRef="jdbclib" />  
    <connectionManager maxPoolSize="50" />  
    <properties.db2.jcc driverType="4"  
        databaseName="DB2A"  
        currentSchema="DSN81210"  
        serverName="localhost"  
        portNumber="38000" />  
</dataSource>  
<authData id="db2user" user="DB2USER" password="{xor}0z1tLz4sLA==" />
```

Having made these changes, we restarted our Liberty JVM sever and ran our Employee application by using the updated configuration. It successfully connected to the data source by using the supplied credentials.

4.5 Transactional support with JDBC

Db2 supports two-phase commit transactions for JDBC type 2 and type 4 modes of connectivity. Both modes of operation can be integrated with the CICS unit-of-work. However, the way in which this process is done is different because the commit processing is handled differently for the two connectivity modes in CICS.

In our Employee application, the updateEmployee() method in the DatabaseOperationsManager class is used to control how updates are committed to the CICS TSQ and the database EMP table. The method provides commit processing logic by using Java Transaction API (JTA) or Connection.commit() processing, based on the setting of the Toggle JTA button in the Employee application.

The following section describes how the Employee sample application is written to provide transactional coordination between CICS and Db2.

JDBC type 2

When JDBC type 2 connectivity is used in CICS, all JDBC commit processing is coordinated by using CICS and the CICS Db2 attachment. The Db2 JCC driver converts any JDBC commit calls or rollback calls into a JCICS commit or a JCICS rollback call, which results in a CICS sync point being taken. As a result, database updates are committed and backed out when the CICS unit-of-work performs a sync point or rolls back.

In addition, if the JDBC application issues a `Connection.commit()` when type 2 connectivity is used, the CICS to perform a sync point that commits the Db2 and CICS updates for the unit-of-work.

The default Db2 behavior when a JDBC data source is used is to set autocommit on, which causes the database manager to perform a commit operation after every SQL statement completes. When used in CICS with type 2 JDBC connectivity, significant extra CICS sync point processing can be generated; therefore, it is set to off by using the `setAutoCommit(false)` method in our sample (see Example 4-11).

Example 4-11 JDBC commit processing in updateEmployee method

```
// Open connection to d/b and execute
conn = ds.getConnection();
conn.setAutoCommit(false);
...
statement.execute();

// Update recoverable CICS TSQ
TSQ tsq = new TSQ();
tsq.setName("DB2LOG");
String msg = String.format("Added %s with last name: %s",
    employee.getEmpNo(), employee.getLastName());
tsq.writeString(msg);

// Commit connection causing syncpoint for type 2
conn.commit();
```

Note: By default, the Liberty data source support (which is now the default configuration in CICS TS V5.4) rolls back on cleanup if no commit was issued before the end of the request. Therefore, if `Connection.commit()` or a `JCICS Task.commit()` is not issued before the end of the Liberty request, the updates to Db2 roll back, which results in an EXEC CICS SYNCPOINT ROLLBACK being issued by CICS. This behavior can be modified by setting the `<dataSource>` attribute `commitOrRollbackOnCleanup="commit"`.

As an alternative to the use of the `Connection.commit()` method, you can also use the JTA to control the commit processing with JDBC type 2 connectivity (see Example 4-12 on page 110). However, the use of JTA is less efficient than the use of the `Connection.commit()` method because it requires another CICS syncpoint to be taken at the start of the `UserTransaction`.

JCBC type 4

When JDBC type 4 connectivity is used in CICS, JDBC commit processing is not coordinated by the CICS recovery manager because the CICS Db2 attachment is not involved. Instead, the JTA should be used to create a Java global transaction, which is coordinated by the Liberty transaction manager. This Java transaction can be used to coordinate the CICS unit-of-work and updates to XA-capable resource managers, such as an XA data source. XA is a two-phase commit protocol that is supported by many databases and application servers, including Db2, Liberty, and CICS.

To use XA support with JDBC type 4 connectivity, it is first necessary to ensure that the data source is configured to use XA. Our `<dataSource>` element was configured with the following attribute to denote it as supporting XA:

```
type="javax.sql.XADataSource"
```

In our Employee application, the `updateEmployee()` method in the `DatabaseOperationsManager` class is used to control how updates are committed. When JTA is enabled, the `updateEmployee()` method instantiates a `UserTransaction` and uses the `UserTransaction.commit()` method to coordinate the JTA transaction, which then controls the subordinate CICS unit-of-work and JDBC connection.

The default Db2 behavior when a JDBC data source is used is to set autocommit on if no JTA transaction is active, which causes the database manager to perform a commit operation after every SQL statement completes. This feature is set to off by using the `setAutoCommit(false)` method in our sample for consistency with the type 2 connectivity scenario.

Our example code is shown in Example 4-12.

Example 4-12 JDBC JTA commit processing in updateEmployee method

```
// Create JTA transaction and syncpoint
UserTransaction utx;
utx = (UserTransaction)
    InitialContext.doLookup("java:comp/UserTransaction");
utx.begin();

// Open connection to d/b and execute
conn = ds.getConnection();
conn.setAutoCommit(false);

...
statement.execute();

// Update recoverable CICS TSQ
TSQ tsq = new TSQ();
tsq.setName("DB2LOG");
String msg = String.format("Added %s with last name: %s",
    employee.getEmpNo(), employee.getLastName());
tsq.writeString(msg);

// Commit JTA transaction and the CICS UOW and XA data source
utx.commit();
```

Note: If a `Connection.commit()` is used in a JDBC application when JDBC type 4 connectivity is used, only the updates to the Db2 data source are committed. Any updates that are part of the CICS unit-of-work must be committed separately.

4.6 Tracing JDBC

To diagnose problems with JDBC connectivity in CICS, it can be useful to use the Db2 JCC driver trace. We used extra properties on our `<dataSource>` `<properties.db2.jcc>` sub-element to activate tracing for data sources that are defined in our CICS Liberty environment, as shown in Example 4-13.

Example 4-13 Enabling JCC trace properties

```
<properties.db2.jcc driverType="4"
    databaseName="DB2A"
```



```
traceDirectory="/var/cicsts/SC8CICS2/wlp/servers/itsowlp1/logs/"
traceFile="jcc.trc"
traceFileAppend="false"
traceLevel="-1"
user="DB2USER"
currentSchema="DSN81210"
password="{xor}0z1tLz4sLA=="
serverName="localhost"
portNumber="38000" />
```

For more information about the available JCC trace properties, see the [Enabling trace for a datasource connection section](#) of the Collecting Data: Tracing with the IBM Data Server Driver for JDBC and SQLJ page of the IBM Support website.



Connecting to IBM MQ by using JMS

In this chapter, we introduce Java Message Service (JMS) and describe how we deployed two sample Java Messaging Service API applications to our CICS Liberty JVM server. The two JMS applications we used were from the CICS sample JMS application, `cics-java-liberty-mq-jms` repository, which is available from the [CICS development organization's section in GitHub](#).

This chapter includes the following topics:

- ▶ 5.1, “Introduction to JMS” on page 114
- ▶ 5.2, “JMS sample application” on page 116
- ▶ 5.3, “Required CICS resources” on page 122
- ▶ 5.4, “Required IBM MQ resources” on page 125
- ▶ 5.5, “Testing the sample applications” on page 126
- ▶ 5.6, “Security” on page 129
- ▶ 5.7, “Transport Layer Security” on page 136

Tip: Support for JMS in CICS differs from the JMS support that was added in the CICS TS V5.2 APAR PI32151. The original support for JMS in CICS was enabled in a CICS OSGi JVM server by adding IBM MQ OSGi JAR files to the `OSGI_BUNDLES` directive in a CICS JVM profile and connectivity to a queue manager was managed by CICS using `MQCONN` resource definitions.

No support was available for message-driven beans in the original support. Also, no support was available for `MQMONITOR` as a resource in the Liberty JVM support.

5.1 Introduction to JMS

In this section, we introduce JMS and message driven beans (MDB) and describe the importance of the Java Naming and Directory Interface (JNDI) when JMS applications are written.

5.1.1 Java Message Service

JMS is an application programming interface (API) that is used for sending and receiving messages between applications. When a Java application uses JMS classes and methods to send and receive messages as described by the Java EE specification for JMS, that application is independent of the underlying messaging services provider; for example, IBM MQ, and IBM WebSphere Application Server. Because of this independence, the underlying message provider can be changed without rewriting any application code.

A JMS non-MDB application interacts with `ConnectionFactory` and `Destination` objects. An instance of a `ConnectionFactory` object includes instance variables that contain the connection properties that are specific to a queue manager (for example, whether server or client bindings, queue manager name, host, or port). A `Destination` object includes the instance variables that are based on the values of the properties of the queue or topic. Both types of objects also include the methods that are required to interact with their respective targets.

5.1.2 Message Driven Beans

An MDB is Java code that is started or triggered when a message arrives on a designated queue or topic. The MDB features a method that is named `onMessage`, which is automatically started by the arrival of a message. The message is passed as a parameter to the `onMessage` method.

5.1.3 Java Naming and Directory Interface

JNDI is a standard interface to a repository, or name space, which allows resources to be located by name. Resources might include simple text values, JDBC data sources, JMS, and objects. Applications can use JNDI to locate resources at run time, which minimizes the coupling between applications and the configuration of the resource that they need.

From a JMS perspective, the resources that are defined to JNDI are `ConnectionFactory`s or `Destinations` (for example, queues or topics). A JMS application uses a JNDI lookup to create a `ConnectionFactory` object. A JMS application also uses a JNDI lookup to create `Destination` objects.

The use of JNDI lookups of JMS resources at runtime means that JMS applications can be written without providing specific names for a queue manager, queues, or topics and without specifying an explicit connection or destination name. That is, a JMS application can use the same application-specific names (known as a *JNDI name* or *alias*) for queue managers, queues, or topics during developing, test, and production. This configuration allows an administrator or deployer to associate these application names with the actual queue manager connection details, queue properties, and so on, at runtime.

Example 5-1 shows a Liberty `server.xml` name space configuration in which IBM MQ is the messaging provider. It defines one JMS connection factory for a local queue manager (LQM) and four JMS connection factories for remote connections to queue managers (Transport Layer Security [TLS] and non TLS). One JMS destination is defined.

Example 5-1 Liberty server.xml

```
<jmsConnectionFactory jndiName="jms/LMQM">
  <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" queueManager="QML1"
transportType="BINDINGS"/>
</jmsConnectionFactory>

<jmsConnectionFactory jndiName="jms/QML1">
  <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostName="mpx1" port="1417"
queueManager="QML1"/>
</jmsConnectionFactory>

<jmsConnectionFactory jndiName="jms/QML2">
  <properties.wmqJms channel="SYSTEM.SSL.SVRCONN" hostName="mpx2" port="1418"
queueManager="QML2"
sslCertStores="keystore.jks"/>
</jmsConnectionFactory>

<jmsConnectionFactory jndiName="jms/QML3">
  <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostName="mpx1" port="1419"
queueManager="QML3"/>
</jmsConnectionFactory>

<jmsConnectionFactory jndiName="jms/QML4">
  <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostName="mpx2" port="1420"
queueManager="QML4"/>
</jmsConnectionFactory>

<jmsQueue id="Queue" jndiName="jms/Queue">
  <properties.wmqJms baseQueueName="SYSTEM.DEFAULT.LOCAL.QUEUE"/>
</jmsQueue>
```

The JMS application can look up a JNDI name `jms/QML1` to instantiate the `ConnectionFactory` and lookup JNDI name `jms/Queue` to instantiate a destination object. The use of a JNDI lookup means that the JMS resources can be changed with no modifications that are required to the application when the runtime environment is changed.

For Liberty, the `jmsConnectionFactory` and `jmsQueue` elements in the `server.xml` binds the resources into the JNDI name space. The JMS application performs a lookup into the name space and the name space information is used to instantiate Java objects whose instance variables and methods provide the means to interact with queue managers and queues.

As shown in Example 5-2 on page 116, the Java object `qcf` is created and its instance variables are populated by starting a context lookup of JNDI name `jms/QML1`. The same applies for the Java object **destination** (a queue) when a context lookup of JNDI name `jms/Queue` is performed.

Example 5-2 JMS JNDI lookup code sample

```
ConnectionFactory queueConnectionFactory = null;
Destination destination = null;
// Lookup the connection factory using string "jms/QML1"
ConnectionFactory queueConnectionFactory = (ConnectionFactory)
    context.lookup("jms/QML1");
Destination destination = (Destination)
    context.lookup("jms/Queue");
```

The Java code in Example 5-2 shows the JNDI lookup of a `ConnectionFactory` and `Destination` object.

After the `ConnectionFactory` objects and any destinations objects are created or instantiated, they can be used by the Java code to connect to the queue manager, establish a session, and to send and receive of messages to queues or publish messages to topics.

5.1.4 Connection pooling

The overhead of establishing a connection to a queue manager can be expensive in terms of resource consumption and time. To help reduce these costs, the Liberty runtime provides for connection pooling. Connection pooling means that the runtime container (that is, Liberty in CICS in our scenario) maintains a pool of connections and an available connection in the pool that is not actively being used. These connections can be reused by a new request, which avoids the cost of creating a connection. Connection pools can be configured to specify the following parameters:

- ▶ Maximum and minimum pool size
- ▶ Connection request timeout period, which is the time that a request waits for a response before ending the request
- ▶ Aged timeout period, which specifies a period that an inactive connection is purged from the pool

5.2 JMS sample application

In this section, we describe the sample application that is deployed to CICS Liberty. The JMS sample application that we used, `cics-java-liberty-mq-jms`, is available on the [cicsdev/cics-java-liberty-mq-jms](https://github.com/cicsdev/cics-java-liberty-mq-jms) page of the GitHub website.

This sample application features two Java classes: `MQJMSDemo` and `MySimpleMDB`.

`MQJMSDemo` is started by using a web browser and accepts a query parameter with one of the following values:

- ▶ `readq`: Receive a message from the test queue.
- ▶ `putq`: Send a message to the test queue.
- ▶ `putmdbq`: Send a message to the MDB queue.
- ▶ `readtsq`: Display the contents of the CICS temporary storage queue.

`MQJMSDemo` is started by entering a URL that is based on the CICS region's host name and the port on which the CICS Liberty JVM server is listening along with the URL path `/jms?test=parameter`; for example, `http://host:port/jmsweb?test=readq` reads a message from the test queue.

MySimpleMDB is started by using the MQJMSDemo application to put a message on the MDB trigger queue; for example, `http://host:port/jmsweb?test=putmdbq`.

The MySimpleMDB application is passed the message as a parameter and writes the contents of the message along with a date and time stamp to a CICS temporary storage queue.

5.2.1 Modifying the JMS sample application

To build and deploy the sample applications, we needed a development environment with the CICS Explorer version 5.4.x with CICS SDK for Java EE and Liberty plug-in installed.

The source for the project is available on the [cicsdev/cics-java-liberty-mq-jms](#) page of the GitHub website.

At the website, we clicked **Clone or download** and then, **Download ZIP**. The master branch of the repository was then downloaded as a .zip file that is named `cics-java-liberty-mq-jms.master.zip` in our download directory.

To modify the application, we imported it into our CICS Explorer workspace by completing the following steps:

1. In our CICS Explorer session, we switched to the Java perspective.
2. We selected **File** on the toolbar.
3. From the drop-down list, we selected the **Import** option.
4. In the Import-Select window, we selected the **Existing Projects into Workspace** option under the General folder.
5. In the Import-Import Project window, we selected the **Select archive file** option.
6. We clicked **Browse** to locate the Download directory where we selected **cics-java-liberty-mq-jms.master.zip**.
7. We clicked **Open** to import the archive file into the workspace.

When complete, our workspace resembled the window that is shown in Figure 5-1 on page 118.

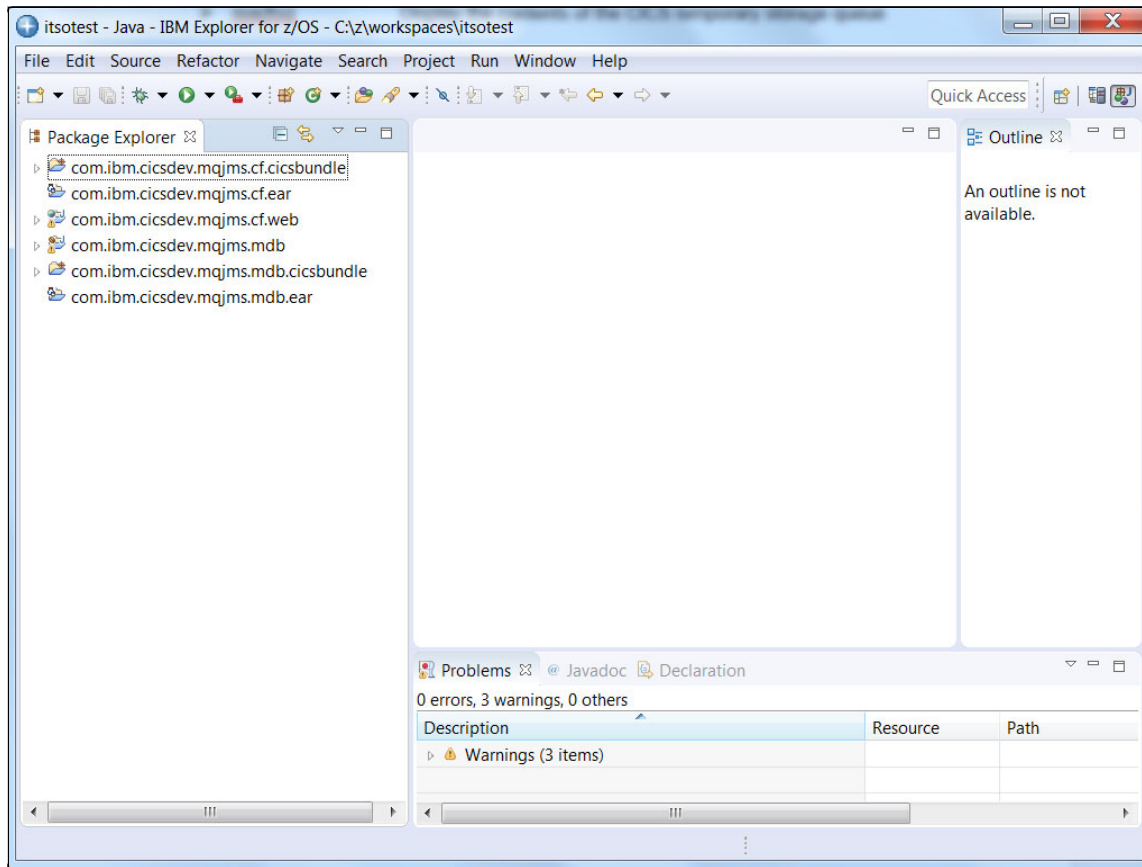


Figure 5-1 Imported JMS sample applications

Before the bundles can be deployed into the Liberty JVM server, the bundles must be changed to reflect the JVM server in which our Liberty JVM server is running. As provided from GitHub, the JVM Server is set to DFHWLP, which is not the name of our JVM Server.

We expanded the two bundle projects and changed the contents of the two CICS bundles files `com.ibm.cicsdev.mqjms.cf.web.warbundle` and `com.ibm.cicsdev.mqjms.mdb.ear.earbundle` so that the `jvmserver` property is set to `IT0SJVM1`, as shown in Figure 5-2.



Figure 5-2 The `com.ibm.cicsdev.mqjms.cf.web.warbundle`

5.2.2 Deploying the JMS sample application

After the bundle is updated with the target JVM Server name, the bundle is ready for deployment.

We switched to the Remote Systems Explorer perspective and complete the following steps:

1. We opened the **Host Connections** tab to create an FTP connection to our z/OS image.
2. After we connected to the host, we switched to the Java EE perspective and selected the `com.ibm.cicsdev.mqjms.cf.cicsbundle` project.
3. We right-clicked and selected the **Export Bundle Project to z/OS UNIX System Filesystem** option.
4. We wanted to export the bundle to the bundle directory that is configured in the CICS region, so we selected the **Export to a specific location in the file system** option.
5. We clicked the **Next** button.
6. In the Export Bundle window, we entered `/var/cicsts/SC8CICS7/ITS0JVM1/bundles` in the field that is next to Parent Directory.
7. We clicked **Finish**.

Note: If the bundle was deployed, the **Clear existing contents of the Bundle directory** option must be selected.

5.2.3 Configuring Liberty for the JMS sample application

The .zip file that was downloaded from GitHub also included a `server.xml` file that we used to configure the Liberty JVM server so that JMS and this application is supported.

We modified the `server.xml` file (see Example 5-3) by completing the following tasks:

- ▶ Set the value of the `wmqJmsClient.rar.location` variable to the location of the IBM MQ resource adapter archive (RAR) file (for example, `wmq.jmsra.rar`). This setting provides runtime access to JMS Java classes and any required executables to the CICS Liberty JVM server.

Note: A resource adapter archive file is provided by a software vendor. It contains the Java code that is required to access vendor's resource. For more information about the resource adapter archive that is provided by IBM MQ, see the [Using the IBM MQ resource adapter page](#) of IBM Knowledge Center.

- ▶ Changed all occurrences of port attribute to the TCP/P port on which the channel initiator task is listening.
- ▶ Changed all occurrences of the `queueManager` attribute to the queue manager subsystem name.

Example 5-3 Customized server.xml file

```
<server description="MQ JMS Liberty sample">
  <featureManager>
    <feature>wmqJmsClient-2.0</feature>
    <feature>mdb-3.2</feature>
    <feature>jndi-1.0</feature>
  </featureManager>
```

```

<variable name="wmqJmsClient.rar.location"
value="/usr/lpp/mqm/V9R0M0/java/jca/wmq.jmsra.rar"/>

<jmsQueueConnectionFactory connectionManagerRef="ConMgrJms"
    jndiname="jms/qcf1">
    <properties.wmqJms channel="WAS.JMS.SVRCONN"
        hostname="localhost"
        port="1414"
        queueManager="M2A1"
        transportType="CLIENT"/>
</jmsQueueConnectionFactory>

<connectionManager id="ConMgrJms" maxPoolSize="20"/>

<jmsActivationSpec
id="com.ibm.cicsdev.mqjms.mdb.ear/com.ibm.cicsdev.mqjms.mdb/MySimpleMDB">
<properties.wmqJms destinationRef="jms/mdbq"
    destinationType="javax.jms.Queue"
    channel="WAS.JMS.SVRCONN"
    hostname="localhost"
    port="1414"
    queueManager="M2A1"
    transportType="CLIENT" />
</jmsActivationSpec>

<jmsQueue id="jms/simpleq" jndiName="jms/simpleq">
    <properties.wmqJms baseQueueName="DEMO.SIMPLEQ" />
</jmsQueue>

<jmsQueue id="jms/mdbq" jndiName="jms/mdbq">
    <properties.wmqJms baseQueueName="DEMO.MDBQUEUE" />
</jmsQueue>
</server>

```

We copied and renamed the file to a location on the host and added an include statement (see Example 5-4) for this file to the server.xml file that is used by the CICS Liberty JVM server.

Example 5-4 Update to the server.xml for adding support JMS

```

<?xml version="1.0" encoding="UTF-8"?><server description="CICS Liberty profile
sample configuration">
<include location="/var/cicsts/SC8CICS7/wlp/servers/itsowlp1/jms.xml"
optional="true"/>

    <!-- Enable features -->
    <featureManager>
        <feature>cicsts:core-1.0</feature>
        <feature>cicsts:defaultApp-1.0</feature>
        <feature>jsp-2.3</feature>

```

5.2.4 Describing the JMS updates to the JVM server profile

The following elements are added to the CICS Liberty JVM server by including `jms.xml` in the `server.xml` configuration file:

- ▶ `featureManager`: The use of JMS and IBM MQ in a CICS Liberty JVM server required the addition of the following features:
 - `wmqJmsClient-2.0`: Adds support for the IBM MQ messaging provider by using JMS 2.0 interfaces.
 - `mdb-3.2`: Adds support for message driven beans.
 - `jndi-1.0`: Adds support for doing JNDI lookups.
- ▶ `wmqJmsClient.rar.location`: Provides the directory location for the IBM MQ provided resource adapter that contains the required code to connect to IBM MQ.

Tip: In our example, IBM MQ was installed on the same image on which we were running. If the IBM MQ file system is not available on your system, follow the instructions that are found on the [Obtaining the IBM MQ Resource Adapter for the WebSphere Application Server Liberty Profile](#) page of the IBM Support website. Install the resource adapter in any directory you choose. This solution is for client connections only.

- ▶ `jmsQueueConnectionFactory`: Binds a `ConnectionFactory` into the JNDI name space.
- ▶ `jmsActivationSpec`: Associates with one or more message-driven beans with the JNDI names the destinations that are used by them to receive messages.
- ▶ `jmsQueue`: Binds a queue object into the JNDI name space.
- ▶ `connectionManager`: Provides a means to manage the connections between the CICS Liberty JVM server and the queue managers. Properties that can be set include the following examples:
 - The maximum and minimum number of connections to be maintained in a common pool of connections.
 - Parameters that are used to remove stale or old connections from the pool.
 - The frequently with which pool maintenance is performed.

5.3 Required CICS resources

The two applications were deployed to the CICS, as described in 5.2.2, “Deploying the JMS sample application” on page 119. Running these applications required the CICS resources that are described in this section.

5.3.1 BUNDLE resources

Each application required a BUNDLE resource. The BUNDLE definition for the MQJMSDemo sample application is shown in Figure 5-3.

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA  ALter Bundle( MQJMSCF )
  Bundle      : MQJMSCF
  Group       : LIBERTY
  DEScription ==>
  Status      ==> Enabled           Enabled | Disabled
  BUndledir   ==> /var/cicsts/SC8CICS7/ITS0JVM1/bundles/com.ibm.cicsdev.mqjm
  (Mixed Case) ==> s.cf.cicsbundle_1.0.0
  ==>
  ==>
  BAsescope   ==>
  (Mixed Case) ==>
  ==>
  ==>
  DEFINITION SIGNATURE
  DEFinetime  : 10/10/17 12:17:40
+  CHANGETime  : 10/10/17 12:24:37
                                           SYSID=SC87 APPLID=SC8CICS7

PF 1 HELP 2 COM 3 END                    6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 5-3 CICS BUNDLE definition for sample application MQJMSDemo

The BUNDLE definition for the MySimpleMDB sample application is shown in Figure 5-4.

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA  ALter Bundle( MQJMSMDB )
  Bundle      : MQJMSMDB
  Group       : LIBERTY
  DEScription ==>
  Status      ==> Enabled           Enabled | Disabled
  BUndledir   ==> /var/cicsts/SC8CICS7/ITS0JVM1/bundles/com.ibm.cicsdev.mqjm
  (Mixed Case) ==> s.mdb.cicsbundle_1.0.0
  ==>
  ==>
  BAsescope   ==>
  (Mixed Case) ==>
  ==>
  ==>
  DEFINITION SIGNATURE
  DEFinetime  : 10/10/17 12:18:24
+  CHANGETime  : 10/10/17 13:36:40
                                           SYSID=SC87 APPLID=SC8CICS7

PF 1 HELP 2 COM 3 END                    6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 5-4 CICS Bundle definition for sample application MySimpleMDB

These resources were installed in to the CICS region SC8CICS7.

5.3.2 URIMAP resource

A URIMAP resource was defined so that the MQJMSDemo application runs under its own transaction identifier, as shown in Figure 5-5.

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA Alter Urimap( ITS0JJMS )
  Urimap      : ITS0JJMS
  Group       : LIBERTY
  Description ==> LIBERTY URIMAP FOR DEFAULT APP
  Status      ==> Enabled          Enabled | Disabled
  Usage       ==> Jvmserver        Server | Client | Pipeline | Atom
                                           | Jvmserver

  UNIVERSAL RESOURCE IDENTIFIER
  Scheme      ==> HTTP             HTTP | HTTPS
  Port        ==> No              No | 1-65535
  Host        ==> *
  Path        ==> jmsweb/*
  (Mixed Case) ==>
  ==>
  ==>
+ OUTBOUND CONNECTION POOLING

                                           SYSID=SC87 APPLID=SC8CICS7

PF 1 HELP 2 COM 3 END                6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 5-5 CICS resource definition for JMS URIMAP (1 of 2)

Scroll forward to show the JJMS transaction (), as shown in Figure 5-6.

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA Alter Urimap( ITS0JJMS )
+ Port        ==> No              No | 1-65535
  Host        ==> *
  Path        ==> jmsweb/*
  (Mixed Case) ==>
  ==>
  ==>
  OUTBOUND CONNECTION POOLING
  Socketclose ==>                0-240000 (HHMMSS)
  ASSOCIATED CICS RESOURCES
  TCpipservice ==>
  Analyzer     ==> No            No | Yes
  Converter    ==>
  Transaction  ==> JJMS
  Program      ==>
+ Pipeline    ==>

                                           SYSID=SC87 APPLID=SC8CICS7

PF 1 HELP 2 COM 3 END                6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 5-6 CICS resource definition for JMS URIMAP (2 of 2)

These resources were installed in to the CICS region.

5.3.3 Transaction resources

Transaction JJMS was defined as alternative way to start program DFHJSTHP, which is an alias for CJSJ (see Figure 5-7).

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA  ALTER TRANSAction( JJMS )
  TRANSAction      : JJMS
  Group            : LIBERTY
  DEScription      ==> LIBERTY JMS Sample application
  PROGram          ==> DFHJSTHP
  TWasize          ==> 00000                      0-32767
  PROFile          ==> DFHCICST
  PArTitionset     ==>
  STAtus           ==> Enabled                    Enabled | Disabled
  PRIMedsize       : 00000                      0-65520
  TASKDATAloc      ==> Any                        Below | Any
  TASKDATAKey      ==> User                       User | Cics
  STOrageclear     ==> No                         No | Yes
  RUNaway          ==> System                     System | 0 | 250-2700000
  SHutdown         ==> Disabled                   Disabled | Enabled
  ISolate          ==> Yes                         Yes | No
  Brexit           ==>
+ REMOTE ATTRIBUTES

                                           SYSID=SC87 APPLID=SC8CICS7

PF 1 HELP 2 COM 3 END                      6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 5-7 CICS resource definition for JMS transaction JMS

After these transaction definitions are installed, they are used to isolate the JMS sample from other applications that are running in CICS Liberty, as shown in Figure 5-8.

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA  ALTER TRANSAction( JMDB )
  TRANSAction      : JMDB
  Group            : LIBERTY
  DEScription      ==> LIBERTY unclassified work
  PROGram          ==> DFHJSTHP
  TWasize          ==> 00000                      0-32767
  PROFile          ==> DFHCICST
  PArTitionset     ==>
  STAtus           ==> Enabled                    Enabled | Disabled
  PRIMedsize       : 00000                      0-65520
  TASKDATAloc      ==> Any                        Below | Any
  TASKDATAKey      ==> User                       User | Cics
  STOrageclear     ==> No                         No | Yes
  RUNaway          ==> System                     System | 0 | 250-2700000
  SHutdown         ==> Disabled                   Disabled | Enabled
  ISolate          ==> Yes                         Yes | No
  Brexit           ==>
+ REMOTE ATTRIBUTES

                                           SYSID=SC87 APPLID=SC8CICS7

PF 1 HELP 2 COM 3 END                      6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 5-8 CICS resource definition for the JMDB JMS transaction

Tip: The transaction for the MDB application is specified by JVM profile directive `-Dcom.ibm.cics.jvmserver.unclassified.tranid=JMDB`.

MDBs run under CJSU because they are unclassified requests (they are not classified as HTTP requests). Therefore, they fall into the category of unclassified threads that need a CICS transaction context.

5.4 Required IBM MQ resources

The following queues are required for the sample applications:

- ▶ Queue DEMO.SIMPLEQ is used by MQJMSDemo
- ▶ Queue DEMO.MDBQUEUE is used by MySimpleMDB

We used IBM MQ Explorer to define these queues.

Tip: IBM MQ Explorer is available as an IBM MQ Support Pac, which can be downloaded from [the MS0T: IBM MQ Explorer page](#) of the IBM Support website. It also is provided as part of the Linux or Windows IBM MQ installation.

5.4.1 Configuring IBM MQ Explorer

We downloaded and installed IBM MQ Explorer. We then started it and configured a connection to the queue manager M2A1 that used port 1414 (the same port that appears in the Liberty configuration file).

5.4.2 Defining the queues

After we were connected to the queue manager, we expanded the folder for M2A1 on 'wstc80(1414)' and right-clicked **Queues**.

We selected **New** and followed the steps to create a local queue that is named DEMO.SIMPLEQ and DEMO.MDBQUEUE.

After the process was completed, two queues were ready for our testing. We used an Explorer filter to limit the display to only the queues whose names began with “DEMO”, as shown in Figure 5-9.

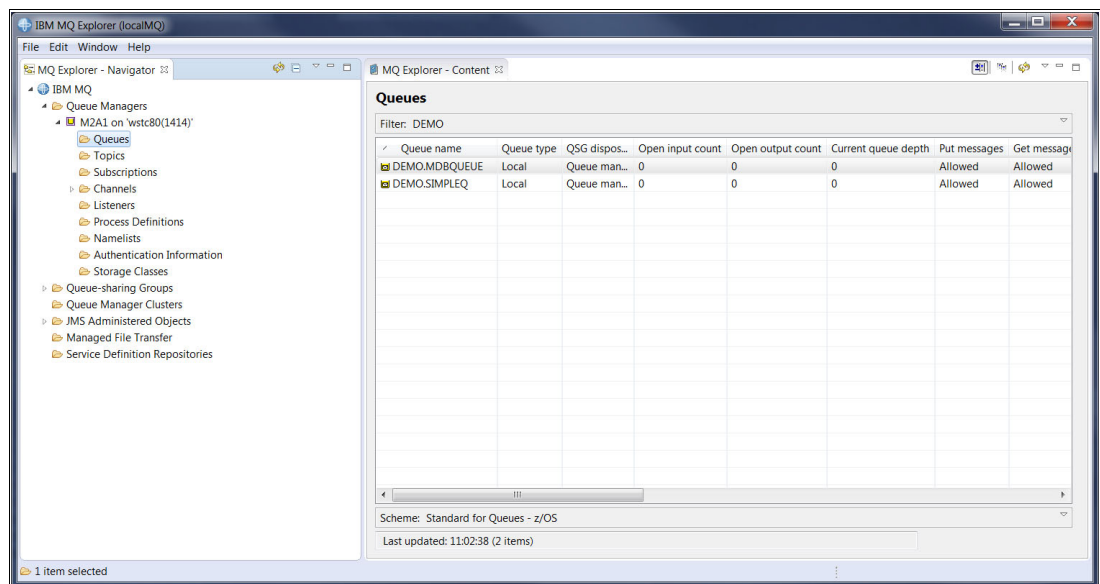


Figure 5-9 Using the IBM MQ Explorer to display information about the DEMO queues

5.5 Testing the sample applications

The next step is to test the applications by using a web browser to start the CICS Liberty applications.

5.5.1 Testing the MQJMSDemo application

To test the MQJMSDemo sample application, we used Firefox and browsed to `http://wstc80:57080/jmsweb/?test=putq` (see Figure 5-10).

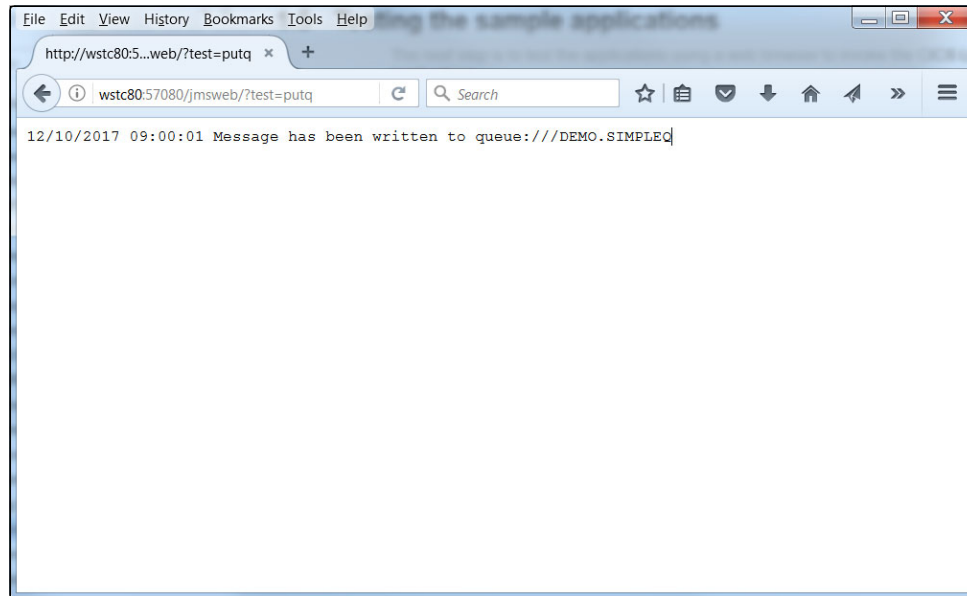


Figure 5-10 Using a browser to put a message on the DEMO.SIMPLEQ queue

For more information, see 5.2, “JMS sample application” on page 116. Clicking **Refresh** in the browser window writes another message to the queue.

Switching to the IBM MQ Explorer session (see Figure 5-11), we browsed the contents of the DEMO.SIMPLEQ queue, which now contained multiple messages.

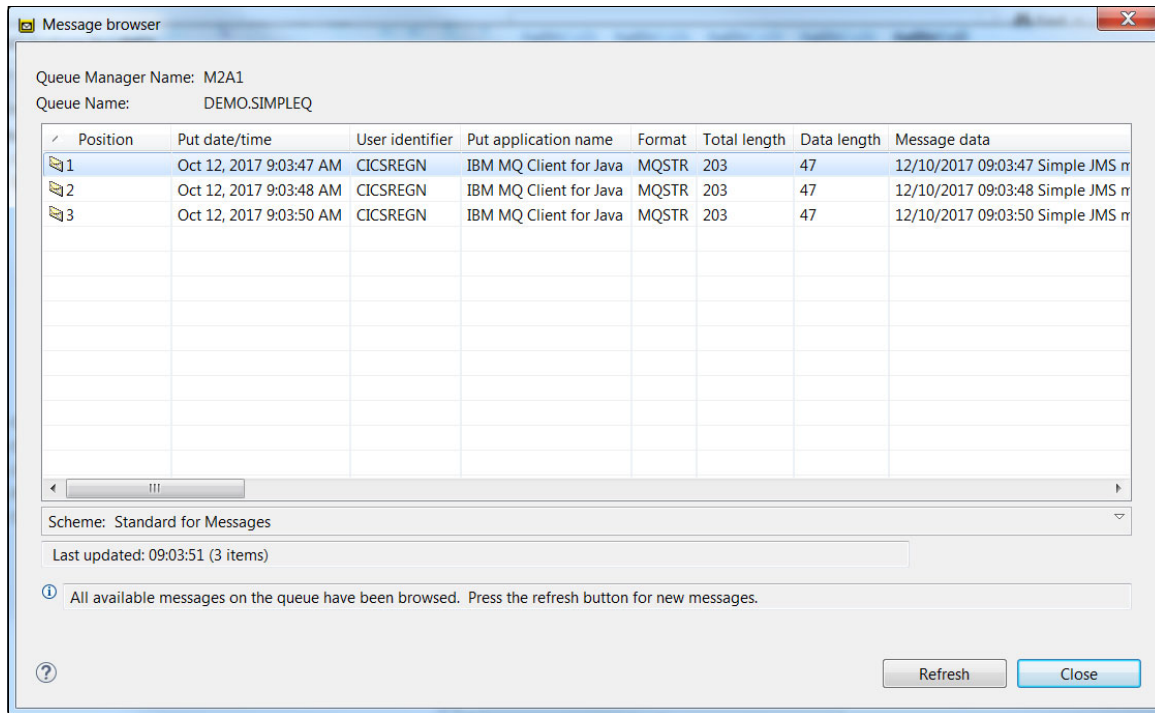


Figure 5-11 Using IBM MQ Explorer to display the messages on the DEMO.SIMPLEQ

Browsing to <http://wstc80:57080/jmsweb/?test=readq> starts the method in the sample application that drains all of the messages from the queue and displays them in the browser, as shown in Figure 5-12.

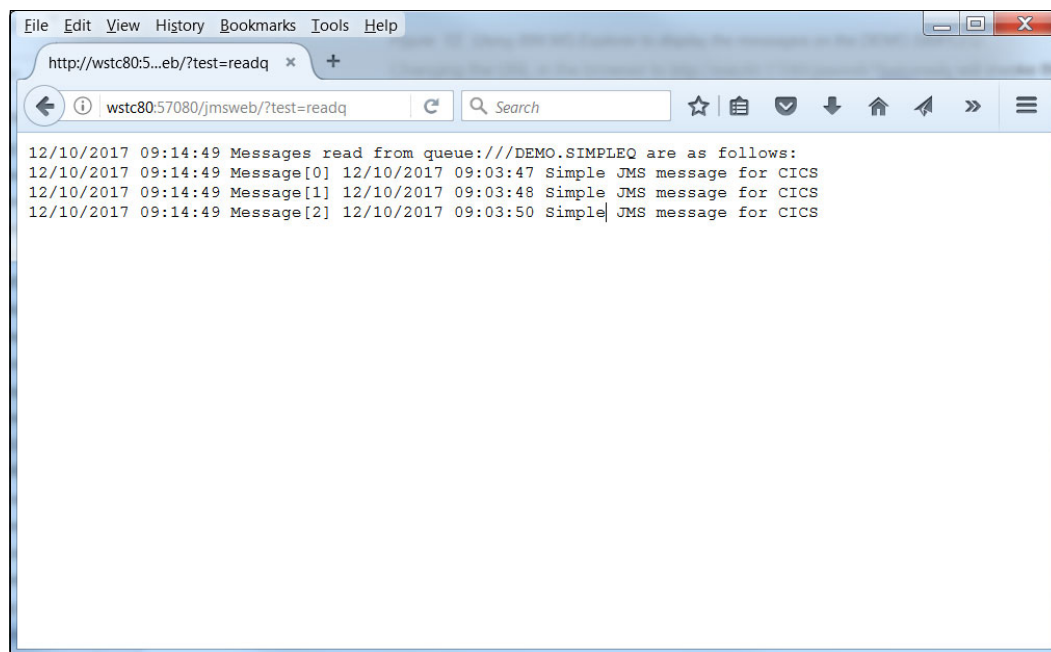


Figure 5-12 Using the browser to get all of the messages from the DEMO.SIMPLEQ

5.5.2 Testing the MySimpleMDB application

To test the MySimpleMDB sample application, we used Firefox (see Figure 5-13) and browsed to `http://wstc80:57080/jmsweb/?test=putmdbq`.

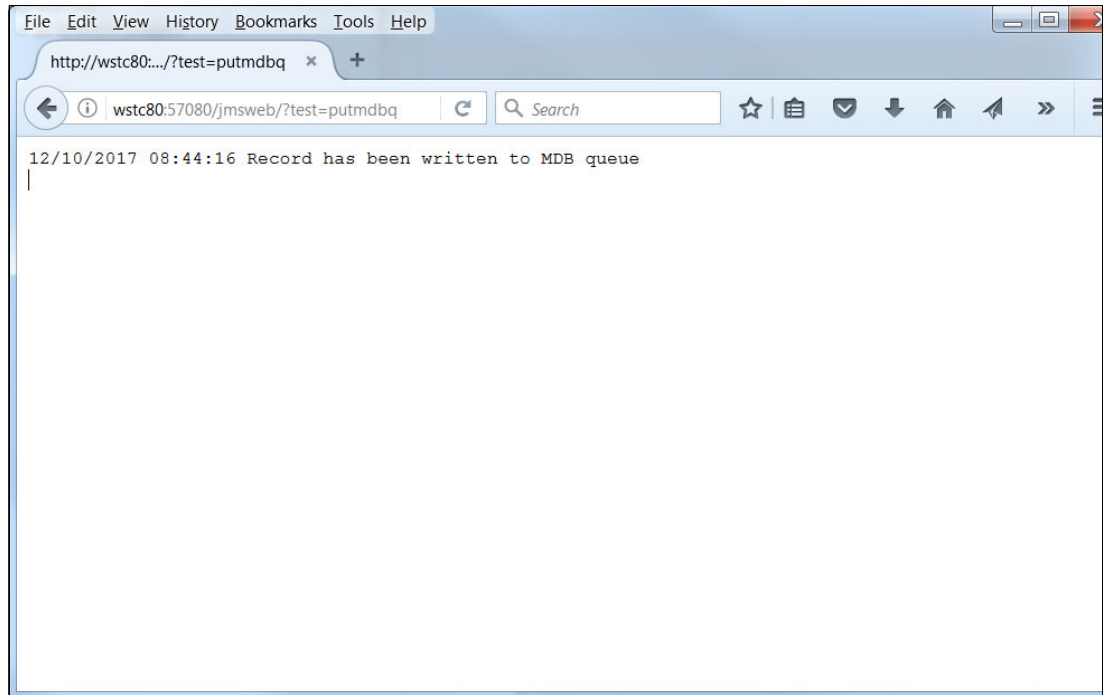


Figure 5-13 Writing a message to the queue that triggers the MDB application

For more information, see 5.2, “JMS sample application” on page 116. Clicking **Refresh** in the browser writes more message to queue.

In a CICS terminal session, we entered transaction CEBR RJMSTSQ to display the contents of the temporary storage queue, as shown in Figure 5-14.

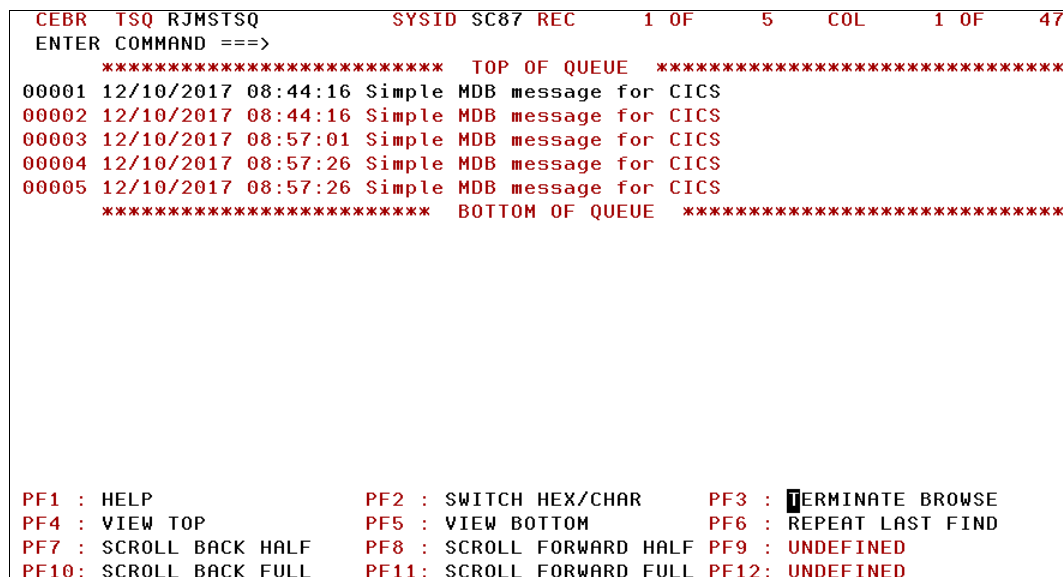


Figure 5-14 Displaying the contents of the default TSQ

5.5.3 Use of the Execution Diagnostic Facility

The CICS provided Execution Diagnostic Facility (EDF) can be used to debug the CICS commands that are issued by the sample applications as they run.

Use of CEDX

We use the CEDX transaction to start an EDF trace on the sample applications.

Entering CEDX JMDB and browsing to the URL in a web browser allowed us to step through the CICS commands in the MySimpleMDB application, as shown in Figure 5-15.

```
TRANSACTION: JMDB PROGRAM: DFHSJTHP TASK: 0000067 APPLID: SC8CICS7 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS WRITEQ TS
  QUEUE ('RJMSTSQ ')
  FROM ('12/10/2017 08:57:26 Simple MDB message for CICS')
  LENGTH (47)
  ITEM (4)
  AUXILIARY
  NOHANDLE

OFFSET:X'1F62BA' LINE: EIBFN=X'0A02'
RESPONSE: NORMAL EIBRESP=0

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: ABEND USER TASK
```

Figure 5-15 EDF display showing the write to TSQ RJMSTSQ in MySimpleMDB

5.6 Security

We wanted to understand the requirements for accessing IBM MQ by using JMS from the CICS Liberty JVM server with IBM MQ security for connections and queues enabled.

5.6.1 RACF resources

We started by defining users, groups, and IBM MQ-related RACF resources (any SAF product is supported). For more information about the RACF classes we used, see [the RACF security classes page](#) of IBM Knowledge Center.

Users and groups

We created the following groups to control access by groups rather than by individual users:

- ▶ MQUSERS for general IBM MQ users who need access to basic IBM MQ functions
- ▶ DEMOUSR for users who run the sample applications
- ▶ MQSTC for the identities of the IBM MQ started tasks and CICS regions

We used the following RACF commands:

- ▶ ADDGROUP MQUSERS
- ▶ ADDGROUP MQSTC
- ▶ ADDGROUP DEMOUSR

- ▶ `CONNECT (JMSUSER,OTHRUSR,CICSREGN) GROUP(MQUSER)`
- ▶ `CONNECT JMSUSER GROUP(DEMOUSR)`
- ▶ `CONNECT M2AG GROUP(MQSTC)`

The following parameters are featured in these commands:

- ▶ M2AG is the identity that is used by the IBM MQ master.
- ▶ Channel initiator address spaces.
- ▶ CICSREGN is the identity that is used by the CICS region.
- ▶ JMSUSER is the JMS test user identity for the sample application.
- ▶ OTHRUSR is an IBM MQ user that should not have the ability to run the sample applications.

MQADMIN resources

We began by enabling IBM MQ RESLEVEL checking by using the **RDEFINE** command to create an RESLEVEL resource for the queue manager in the MQADMIN RACF class, as shown in the following example:

```
RDEFINE MQADMIN M2A1.YES.SUBSYS.SECURITY OWNER(SYS1)
RDEFINE MQADMIN RESLEVEL OWNER(SYS1)
```

Protecting the latter resource means that, different identities (for example, task or address space) are used for authorization depending on the type of connection.

MQCONN resources

We defined an MQCONN resource for the channel initiator task so it can connect to the master task, as shown in the following example:

```
RDEFINE MQCONN M2A1.CHIN CLASS(MQCONN) UACC(NONE)
PERMIT M2A1.CHIN CLASS(MQCONN) RESET
PERMIT M2A1.CHIN CLASS(MQCONN) ID(MQSTC) ACCESS(READ)
```

MQQUEUE resources

We defined a set of MQQUEUE resources for the system-required queues and set of generic profiles for the JMS queues, as shown in the following example:

```
RDEFINE MQQUEUE M2A1.DEMO.** OWNER(SYS1)
PERMIT M2A1.DEMO.** CLASS(MQQUEUE) RESET
PERMIT M2A1.DEMO.** CLASS(MQQUEUE) ID(DEMOUSR,MQSTC) ACC(UPDATE)
```

All of these resource changes were activated by using a **SETROPTS REFRESH TSO** command to rebuild the RACF in-storage profiles, as shown in the following example:

```
SETROPTS RACLIST(MQADMIN,MQQUEUE,MQCONN) REFRESH
```

The use of an **MVS modify** command refreshes the queue managers that are in storage security cache, as shown in the following example:

```
-M2A1 REFRESH SECURITY(*)
```

5.6.2 JMS security scenarios

We tested the following security scenarios regarding the MQJMSDemo application (the MySimpleMDB application did not interact directly with any IBM MQ resources):

- ▶ No security
- ▶ Application managed security
- ▶ Container managed security

No security

The first scenario was the simplest. The MQJMSDemo sample application and the server.xml did not feature any provisions for providing a user identity or password. Therefore, we ran it as is and used the IBM MQ Explorer to identify the user's authority was used to put the messages on the queue.

When we ran our initial test, the user identity that was associated with the messages was the CICS region identity CICSREGN, as shown in Figure 5-16.

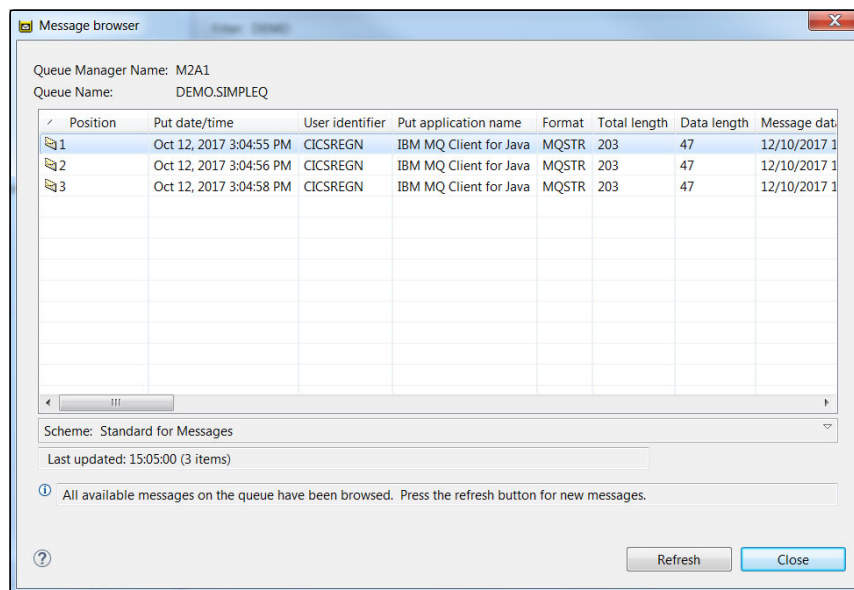


Figure 5-16 No security

Next, we created an IBM MQ channel authentication record to map the CICS region's identity to another identity (for example, JMSUSER), as shown in Figure 5-17.

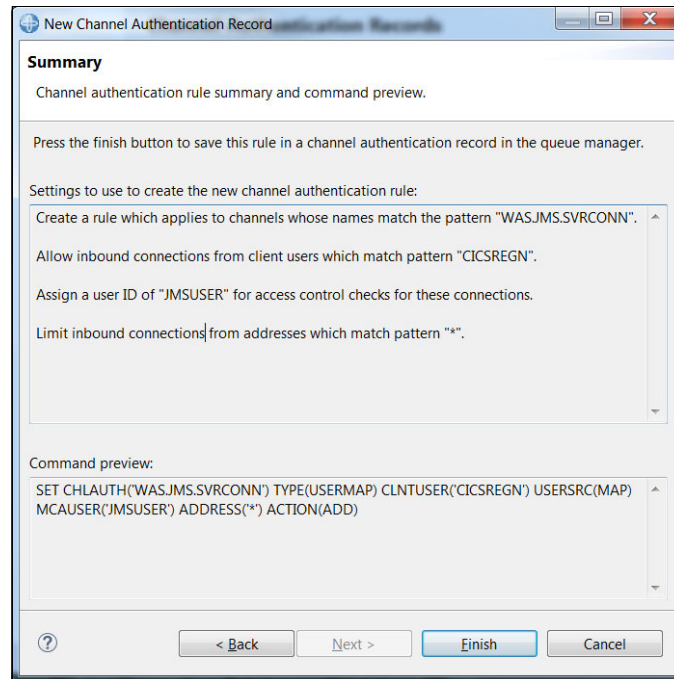


Figure 5-17 Channel authentication record

When we ran our test again, the user identity that was associated with the messages was the identity that was assigned by the channel authentication rule (JMSUSER), as shown in Figure 5-18.

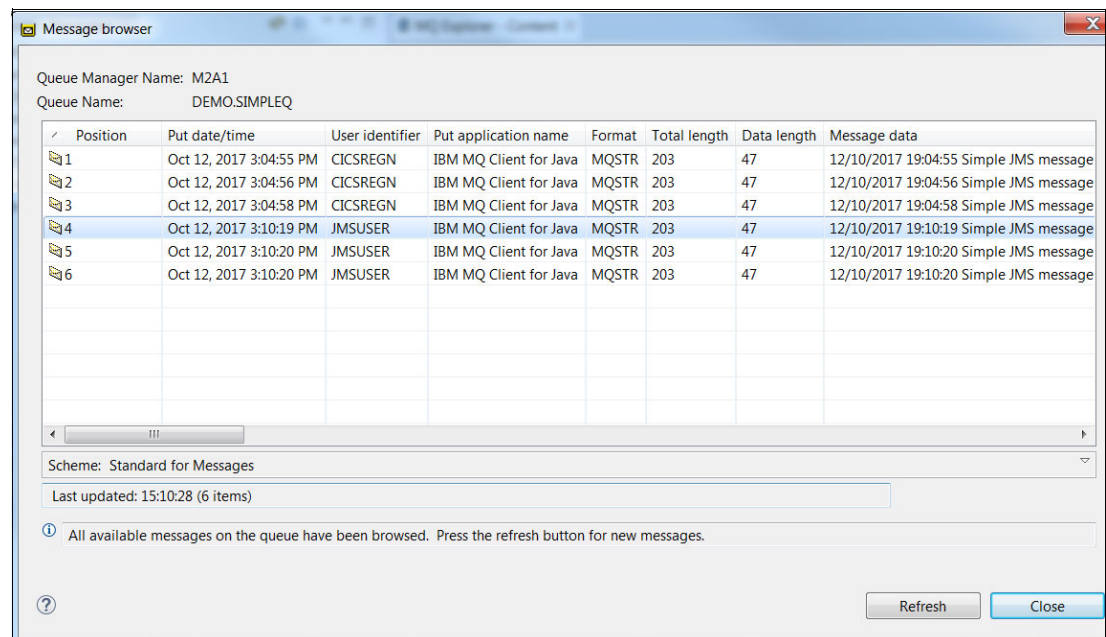


Figure 5-18 Message browser

Application managed security

For application managed security, we modified the createContext method in the Java code in MQJMSDemo by adding user name and password fields.

The first scenario that we tested was with a user with access to IBM MQ and the sample application (for example, JMSUSER), as shown in the following example:

```
String userName = "JMSUSER";  
String password = "JMSUSER";  
JMSContext context = qcf.createContext(userName,password);
```

Before we started testing the modified application for application managed security, we removed the channel authentication record and cleared the queue.

Tip: We used MVS modify command **-M2A1 REFRESH QMGR TYPE(CONFIGEV) OBJECT(CHLAUTH)** to refresh the channel authentication records in the queue manager.

We started `http://wstc80:557080/jmsweb/?test=putq` several times and then used the IBM MQ Explorer to display the User identifier of the messages in the DEMO.SIMPLEQ. As expected, it was set to JMSUSER, as shown in Figure 5-19.

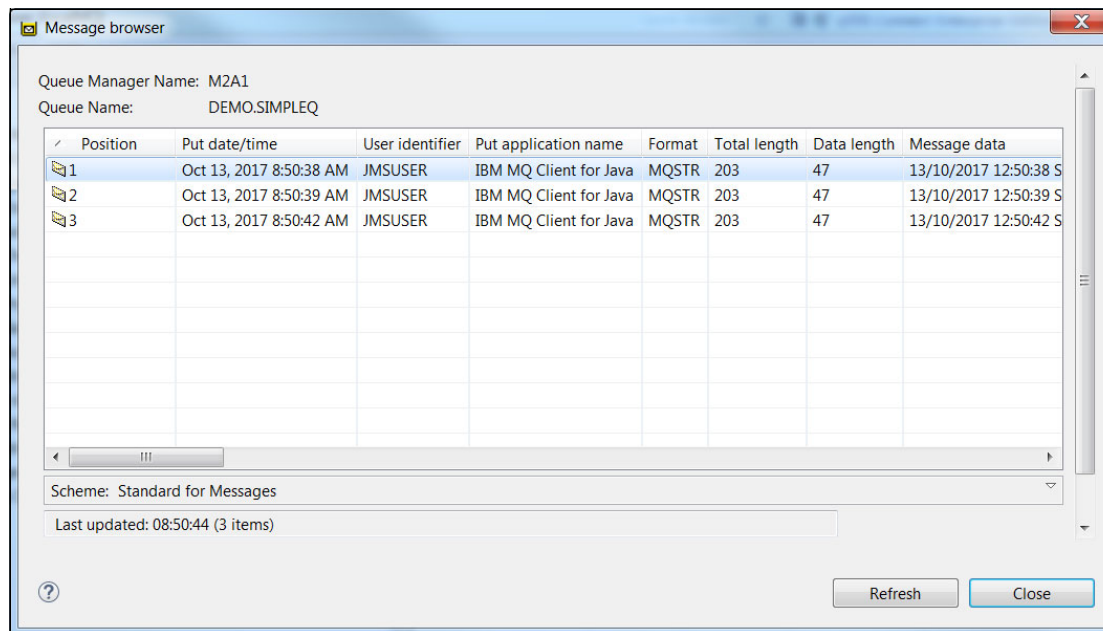


Figure 5-19 Contents of DEMO.SIMPLEQ with JMS application security

Next, we tested with a user with access to IBM MQ (for example, OTHRUSR) but with no access to the DEMO.SIMPLEQ queue, as shown in the following example:

```
String userName = "OTHRUSR";  
String password = "OTHRUSR";
```

We started `http://wstc80:557080/jmsweb/?test=putq`, which failed with the following message:

```
Error 500: javax.servlet.ServletException: ERROR on JMS send JMSWMQ2007: Failed  
to send a message to destination 'DEMO.SIMPLEQ'.
```

The z/OS SYSLOG included these messages, which indicated that this user did not have access to the DEMO.SIMPLEQ queue, as shown in Example 5-5.

Example 5-5 RACF messages displayed when there is no access to the DEMO queues

```
ICH408I USER(OTHRUSR ) GROUP(SYS1 ) NAME(Non JMS User)
M2A1.DEMO.SIMPLEQ CL(MQQUEUE )
INSUFFICIENT ACCESS AUTHORITY
FROM M2A1.DEMO.** (G)
ACCESS INTENT(UPDATE ) ACCESS ALLOWED(NONE)
```

Container-managed security

For container-managed security, we modified the sample application. As before, we began testing by clearing the queue. Also, to verify that we were using the intended container manager security values, we left the code in the sample application that set the user to OTHRUSR.

The first modification to the sample application was to add a resource injection annotation for the qcf ConnectionFactory, as shown in Example 5-6.

Example 5-6 Resource annotation

```
/** JMS connection factory */
@Resource(authenticationType=AuthenticationType.CONTAINER`,name="jms/qcf1")
private static ConnectionFactory qcf;
```

Adding the resource injection meant that the context lookup of the JMS ConnectionFactory was no longer needed; therefore, this code was removed by making it a comment, as shown in Example 5-7.

Example 5-7 Removing context lookup for a ConnectionFactory

```
// JNDI lookups for all the JNDI strings in this test
try {
    InitialContext ctx = new InitialContext();
    //qcf = (ConnectionFactory) ctx.lookup(JMS_CF1);
    simpleq = (Queue) ctx.lookup(JMS_SIMPLEQ);
    mdbq = (Queue) ctx.lookup(JMS_MDBQ);
} catch (NamingException ne) {
    errmsg = " ERROR: On JNDI lookup in servlet initialisation ";
    throw new ServletException(errmsg, ne);
}
```

The server.xml file needed to be updated to include an authData element with container user identity and password. Also, the jmsConnectionFactory needed to be updated with a containAuthDataRef, which identified the JMS authData element, as shown in Example 5-8.

Example 5-8 server.xml updates for container managed security

```
<jmsConnectionFactory connectionManagerRef="ConMgrJms"
    containerAuthDataRef="jmsAuth"
    jndiname="jms/qcf1">
    <properties.wmqJms channel="WAS.JMS.SVRCONN"
        hostname="localhost"
        port="1414"
```



```

        queueManager="M2A1"
        transportType="CLIENT"/>
</jmsConnectionFactory>

<authData id="jmsAuth" user="JMSUSER" password="JMSUSER"/>

```

We started `http://wstc80:557080/jmsweb/?test=putq` several times and used the IBM MQ Explorer to display the User identifier of the messages in the DEMO.SIMPLEQ. As expected, it was set to JMSUSER, as shown in Figure 5-20.

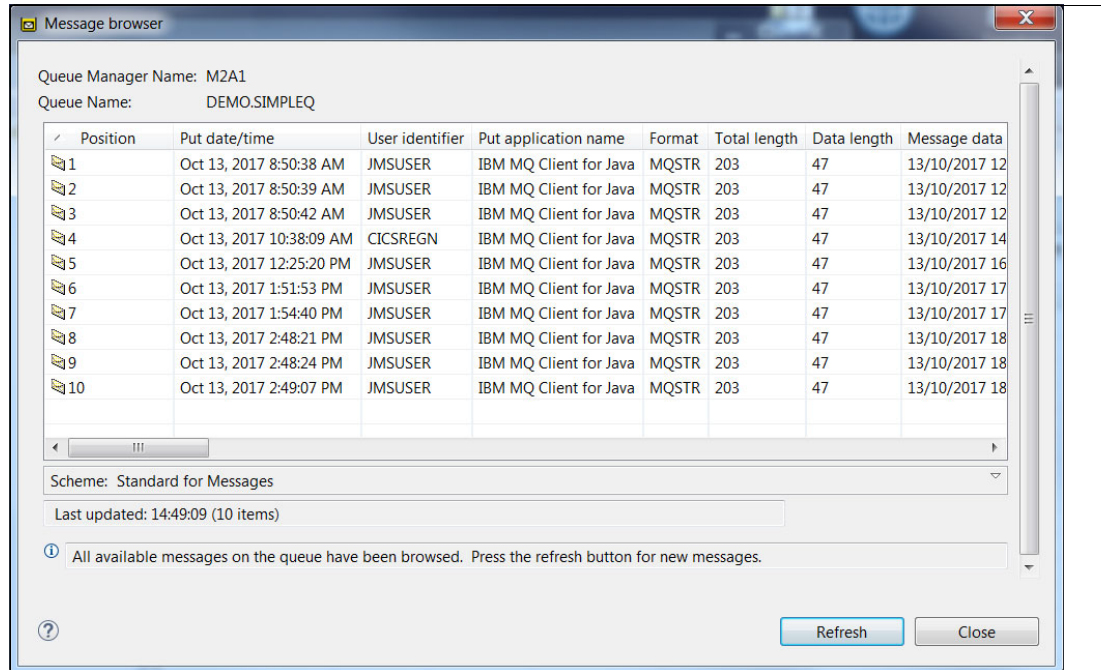


Figure 5-20 IBM MQ Explorer list showing messages created with a container managed identity

5.6.3 Summary

We tested with no special provision for security and observed that the CICS region's RACF identity was used to put messages on the queue. Next, we tested the application managed security, which means that the application provides the security information. Finally, we tested container managed security, which means that the runtime environment (for example, Liberty) provides the security information.

5.7 Transport Layer Security

Next, we wanted to enable Transport Layer Security (TLS) between the CICS Liberty JVM server and the queue manager. This testing required defining more RACF resources and making some other configuration changes to `server.xml`.

5.7.1 RACF resources

We started by defining the required digital certificates.

Certificate authority certificate

We defined a single certificate authority certificate, which is used to sign all of the personal certificates we use during our testing.

The RACF commands that are used to define this certificate are shown in Example 5-9.

Example 5-9 Commands to generate a certificate authority certificate

```
RACDCERT CERTAUTH GENCERT
      SUBJECTSDN(CN('MQ CA') OU('ITSO') O('IBM') C('US'))
      WITHLABEL('MQ CA')
      NOTAFTER(DATE(2020/12/31))
```

Personal certificates

We used the IBM MQ CA certificate to create and signed two personal certificates, one for the channel initiator task and one for the CICS Liberty JVM server.

We used the commands that are shown in Example 5-10 to define these certificates.

Example 5-10 Commands to create and sign personal certificates

```
RACDCERT ID(M2AG) GENCERT
      SUBJECTSDN(CN('MQ CHIN') OU('ITSO') O('IBM') C('US'))
      WITHLABEL('MQ CHIN') SIGNWITH(CERTAUTH LABEL('MQ CA'))
      NOTAFTER(DATE(2020/12/31))

RACDCERT ID(CICSREGN) GENCERT
      SUBJECTSDN(CN('CICSREGN') OU('ITSO') O('IBM') C('US'))
      WITHLABEL('CICSREGN') SIGNWITH(CERTAUTH LABEL('MQ CA'))
      NOTAFTER(DATE(2020/12/31))
```

MQ2G is the RACF authority that is used by the IBM MQ channel initiator task and CICSREGN is the RACF authority of the CICS region.

Key rings

Next, we used the commands that are shown in Example 5-11 to define the key rings and connect the CA and personal certificates to the respective key rings.

Example 5-11 Commands to create key rings and connect certificates to the key rings

```
RACDCERT ID(M2AG)
  ADDRING(MQCHIN.KeyRing)

RACDCERT ID(M2AG)
  CONNECT(RING(MQCHIN.KeyRing)
  LABEL('MQ CA') CERTAUTH)

RACDCERT ID(M2AG)
  CONNECT(RING(MQCHIN.KeyRing)
  LABEL('MQ CHIN') DEFAULT)

RACDCERT ID(CICSREGN)
  ADDRING(CICS.KeyRing)

RACDCERT ID(CICSREGN)
  CONNECT(RING(CICS.KeyRing)
  LABEL('MQ CA') CERTAUTH )

RACDCERT ID(CICSREGN)
  CONNECT(RING(CICS.KeyRing)
  LABEL('CICSREGN') DEFAULT)
```

We changed the queue manager properties to add support for TLS by using the new key ring and personal certificate, as shown in Figure 5-21.

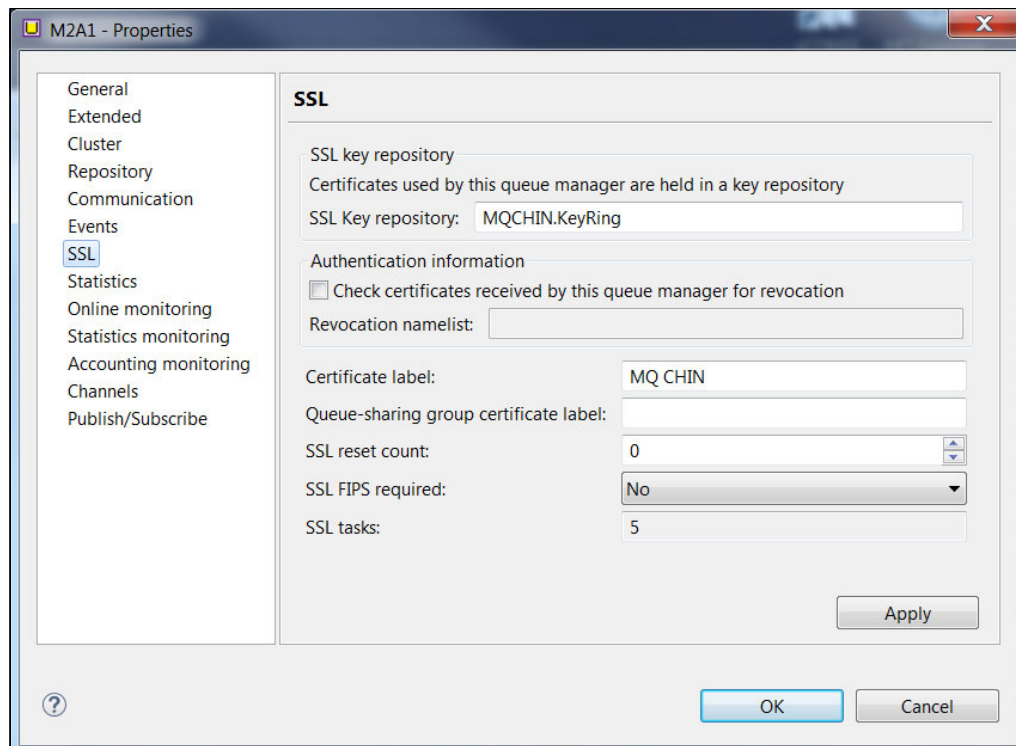


Figure 5-21 Configuring support for TLS in a queue manager

We also configured the cipher on the channel to be used for TLS, as shown in Figure 5-22.

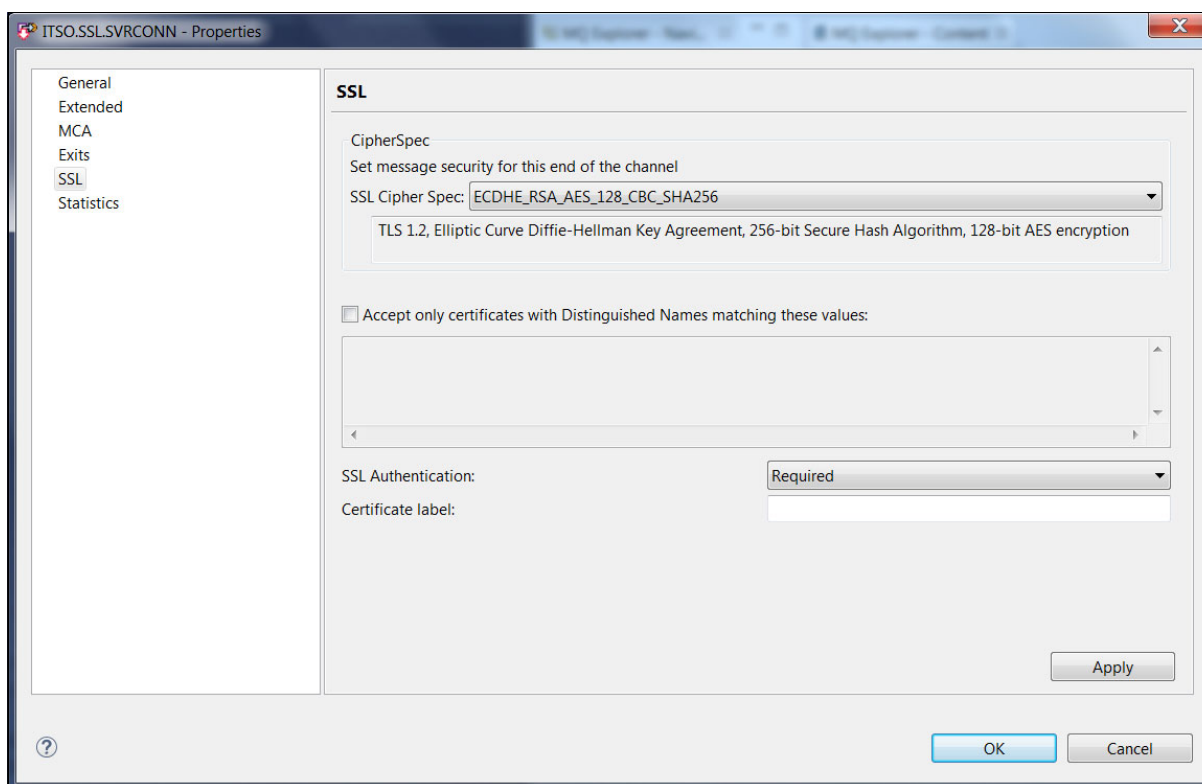


Figure 5-22 Specify a cipher on a channel

We added a keystore for a RACF key ring and encryption cipher information to the `server.xml` file, as shown in Example 5-12.

Example 5-12 `server.xml` updates

```
<ssl id="jmsSSLConfig"
  keyStoreRef="defaultKeyStore"
  trustStoreRef="defaultKeyStore"
  sslProtocol="TLSv1.2"
  enabledCiphers="SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256"
  clientKeyAlias="CICSREGN"/>

<keyStore id="defaultKeyStore"
  location="safkeyring:///CICS.KeyRing"
  password="password" type="JCERACFKS"
  fileBased="false" readOnly="true" />

<jmsConnectionFactory connectionManagerRef="ConMgrJms"
  containerAuthDataRef="jmsAuth"
  jndiname="jms/qcf1">

  <properties.wmqJms channel="ITS0.SSL.SVRCONN"
    hostName="localhost"
    port="1414"
    queueManager="M2A1"
    transportType="CLIENT"
    sslcipherSuite="SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256"/>
```

```

        </jmsConnectionFactory>
    "
    <authData id="jmsAuth" user="JMSUSER" password="JMSUSER"/>

```

Note: The `clientAuthenticationSupported` was set to true in our example; however, it was not necessary because our channel did not provide a client certificate label. If a channel did provide a client certificate, that certificate allows mutual authentication to occur.

The `clientKeyAlias` matched the label of the personal certificate connect to the CICS region's JMS key ring, `CICS.KeyRing`.

After we ran a test, we ran an IBM MQ **DISPLAY** command to display the `SSLCERTU` and `MCAUSER` values on the channel, as shown in Example 5-13.

Example 5-13 Output of a IBM MQ DISPLAY CHSTATUS commands

```

-M2A1 DISPLAY CHSTATUS(ITS0.*) MCAUSER SSLCERTU
CSQM293I -M2A1 CSQMDRTC 1 CHSTATUS FOUND MATCHING REQUEST CRITERIA
CSQM201I -M2A1 CSQMDRTC DISPLAY CHSTATUS DETAILS 634
CHSTATUS(ITS0.JMS.SVRCONN)
CHLDISP(PRIVATE)
CONNNAME(127.0.0.1)
CURRENT
CHLTYPE(SVRCONN)
STATUS(RUNNING)
SUBSTATE(RECEIVE)
STOPREQ(NO)
RAPPLTAG(IBM MQ Client for Java)
SSLCERTU(CICSREGN)
MCAUSER(JMSUSER)
END CHSTATUS DETAILS
CSQ9022I -M2A1 CSQMDRTC ' DISPLAY CHSTATUS' NORMAL COMPLETION

```

This presence of the user identity that is associated with the CICS Liberty's keyring default certificate indicated that a successful TLS handshake was performed between the CICS Liberty JVM server and IBM MQ and that the traffic between the two was being encrypted.

5.7.2 TLS debugging hints and tips

During our testing, we added the Liberty logging element to the `server.xml` to generate security and TLS trace records, as shown in Example 5-14.

Example 5-14 Updating server.xml

```

<logging
    traceSpecification="com.ibm.ws.security.*=all:SSLChannel=all:SSL=all"
    traceFileName="trace.log"
    maxFileSize="20"
    maxFiles="10"
    traceFormat="BASIC

```

A Java directive element was added to the CICS JVM profile ITSOJVM1 to generate JSSE trace records to the STDOUT output location, as shown in Example 5-15.

Example 5-15 Updating ITSOJVM1 (JVM Server profile)

```
-Djavax.net.debug=ssl
```



Configuring Transport Layer Security support

In this chapter, we describe how to configure Secure Sockets Layer (SSL), which is now known as Transport Layer Security (TLS), with a CICS Liberty JVM server. We also describe how to configure TLS client and server authentication scenarios and provide information on how to analyze the use of the IBM z14 cryptographic hardware when TLS is used.

This chapter includes the following topics:

- ▶ 6.1, “JSSE and JCE” on page 142
- ▶ 6.2, “TLS server authentication by using a Java keystore” on page 143
- ▶ 6.3, “TLS server authentication by using a RACF key ring” on page 149
- ▶ 6.4, “TLS client authentication” on page 152
- ▶ 6.5, “Hints and tips when using TLS” on page 156
- ▶ 6.6, “Using cryptographic hardware with JSSE” on page 164

6.1 JSSE and JCE

TLS support in Java uses the underlying Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE) frameworks that are provided as part of Java Standard Edition (Java SE), as shown in Figure 6-1. This support can be integrated with the RACF security registry to control access to digital certificates that are stored in RACF.

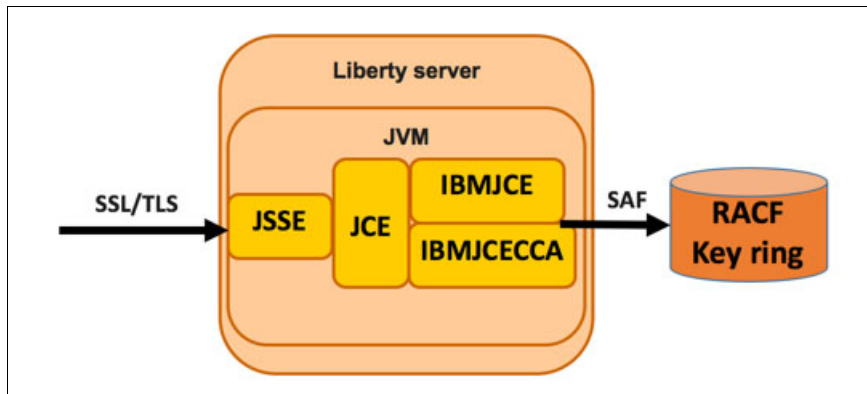


Figure 6-1 TLS support in Java that uses JCE and JSSE

Java Secure Socket Extension

The JSSE API provides a framework and Java implementation of the SSL and TLS protocols as used by Liberty HTTPS support. The JSSE API is provided in the `javax.net` and `javax.net.ssl` packages in Java SE.

Note: IBMJSSE is IBM's Java implementation of TLS and so does not use the facilities of System SSL as used by CICS web support or AT-TLS as provided by IBM Communications Server. Therefore, Liberty in CICS does not use the CICS sockets domain or CICS TCPIP SERVICE resources to configure TLS. All configuration is performed by using the JVM profile and the Liberty server configuration file (`server.xml`).

Java Cryptography Extension

JCE is a standard extension to the Java Platform that provides the underlying implementation for cryptographic services, including encryption, key generation, and Message Authentication Codes (MAC). The IBM Java SDK for z/OS includes the following JCE providers:

- ▶ IBMJCE
- ▶ IBMJCECCA

The use of the JCE provider is controlled by editing the list of security providers. The default provider is IBMJCE and was used in our initial usage scenarios. For more information about how to configure the IBMJCECCA provider to use ICSF to drive the IBM Crypto Express cards, see 6.6, “Using cryptographic hardware with JSSE” on page 164.

6.1.1 Updating the JCE policy files

Because of import regulations in some countries, the strength of certain ciphers is restricted in the default IBMJCE configuration. Therefore, the first step in preparing our system to use TLS is to remove the default restriction on usage of strong ciphers in the JCE policy files.

Note: It is the user's responsibility to verify that this action is permissible under local regulations.

By default, the JCE policy files are in the `$JAVA_HOME/lib/security` directory. The unrestricted versions can be found in `$JAVA_HOME/demo/jce/policy-files/unrestricted/`. The latest unrestricted files are available for download from the [Unrestricted SDK JCE policy files website](#) (login required).

For more information about support for cipher suites with the IBM SDK for Java on z/OS, [see IBM Knowledge Center](#).

Note: If the unrestricted policy files are not installed on your system, stronger ciphers, such as AES_256, fail to operate. In our system, we saw the following exception when the unrestricted policy files were not available:

```
FFDC1015I: An FFDC Incident has been created:
"java.lang.IllegalArgumentException: Cannot support
SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA with currently installed providers
com.ibm.ws.channel.ssl.internal.SSLConnectionLink 238" at ...
```

Installing the policy files

We added the following line to our CICS JVM server profile to activate the IBM supplied unrestricted policy files. Then, we restarted our JVM server:

```
-Dcom.ibm.security.jurisdictionPolicyDir=/usr/lpp/java/
J8.0_64/demo/jce/policy-files/unrestricted
```

6.2 TLS server authentication by using a Java keystore

The default JVM server autoconfigures behavior for TLS in CICS Liberty is to use a Java keystore (JKS) with only server authentication. JKSs are stored in the zFS file system by using files with a `.jks` extension. Because Java keystores are easy to configure, they are useful for initial TLS connectivity testing.

To configure TLS with Liberty, you must first add the `ssl-1.0` feature to the `server.xml` and then, use the `ssl` and `keyStore` elements to define the required settings. These settings are dynamically created if you set the CICS `com.ibm.cics.jvmserver.wlp.server.https.port` system property in the JVM profile when JVM server autoconfigure is used.

In our CICS region, we started by adding the `com.ibm.cics.jvmserver.wlp.server.https.port` property to our JVM profile, as shown in Example 6-1. Then, we restarted the JVM server.

Example 6-1 `com.ibm.cics.jvmserver.wlp.server.https.port` system property

```
-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true
-Dcom.ibm.cics.jvmserver.wlp.server.host=wtsc80
-Dcom.ibm.cics.jvmserver.wlp.server.http.port=57080
-Dcom.ibm.cics.jvmserver.wlp.server.https.port=57443
```

On restart, the JVM server autoconfigure function created the necessary TLS-related elements in the `server.xml`, which caused Liberty to create its own JKS-based keystore and self-signed certificate for use by the HTTPS endpoint.

Our server.xml configuration file now contained the following new entries, as shown in Example 6-2:

- ▶ ssl-1.0: This entry is the SSL feature, and must be added to the feature manager list.
- ▶ httpEndpoint: This entry is the HTTP endpoint that is used to define the attributes of the HTTP and HTTPS listening ports.
- ▶ ssl: This entry is the element that defines the SSL/TLS protocol that is used and the associated keystore.
- ▶ keyStore: This entry is the key store that holds the certificates. The type defaults to a JKS. Although the location defaults to \${server.output.dir}/resources/security/key.jks in zFS, it can be configured to use a RACF key ring. For more information, see 6.3, “TLS server authentication by using a RACF key ring” on page 149. If a trustStoreRef attribute is not defined on the ssl element, the signing certificates also are assumed to be in this keystore.

Example 6-2 SSL support

```
<featureManager>
  <feature>cicsts:core-1.0</feature>
  <feature>jsp-2.3</feature>
  <feature>ssl-1.0</feature>
</featureManager>
<httpEndpoint id="defaultHttpEndpoint"
  host="wtsc80"
  httpPort="57080"
  httpsPort="57443" />

<ssl id="defaultSSLConfig"
  keyStoreRef="defaultKeyStore"
  sslProtocol="TLS" />
<keyStore id="defaultKeyStore"
  password="defaultPassword" />
```

For more information about the TLS configuration attributes for Liberty, see [the Enabling SSL communication in Liberty page](#) of IBM Knowledge Center.

On startup, the Liberty messages.log now reports that the defaultHttpEndpoint-ssl endpoint started with message CWWKO0219I, as shown in Example 6-3.

Example 6-3 HTTP endpoint started

```
CWWKO0219I: TCP Channel defaultHttpEndpoint-ssl has been started and is now
listening for requests on host WTSC80.CPOLAB.IBM.COM   (IPv4: 9.76.61.131) port
57443.
```

We then connected our web browser to the Liberty default landing page by using the following URL. We also specified the HTTPS protocol and the port 57443:

https://wtsc80:57443/

Note: If your browser cannot connect to the HTTPS endpoint in Liberty at this stage, you might need to modify the supported TLS protocols because the supplied default sslProtocol="TLS" restricts support to only TLS v1.0. Setting sslProtocol="SSL" allows any TLS version to be negotiated. For more information, see 6.5.5, “Controlling the TLS version” on page 160.

On loading this page, our Mozilla web browser shows the warning Your connection is not secure, as shown in Figure 6-2.

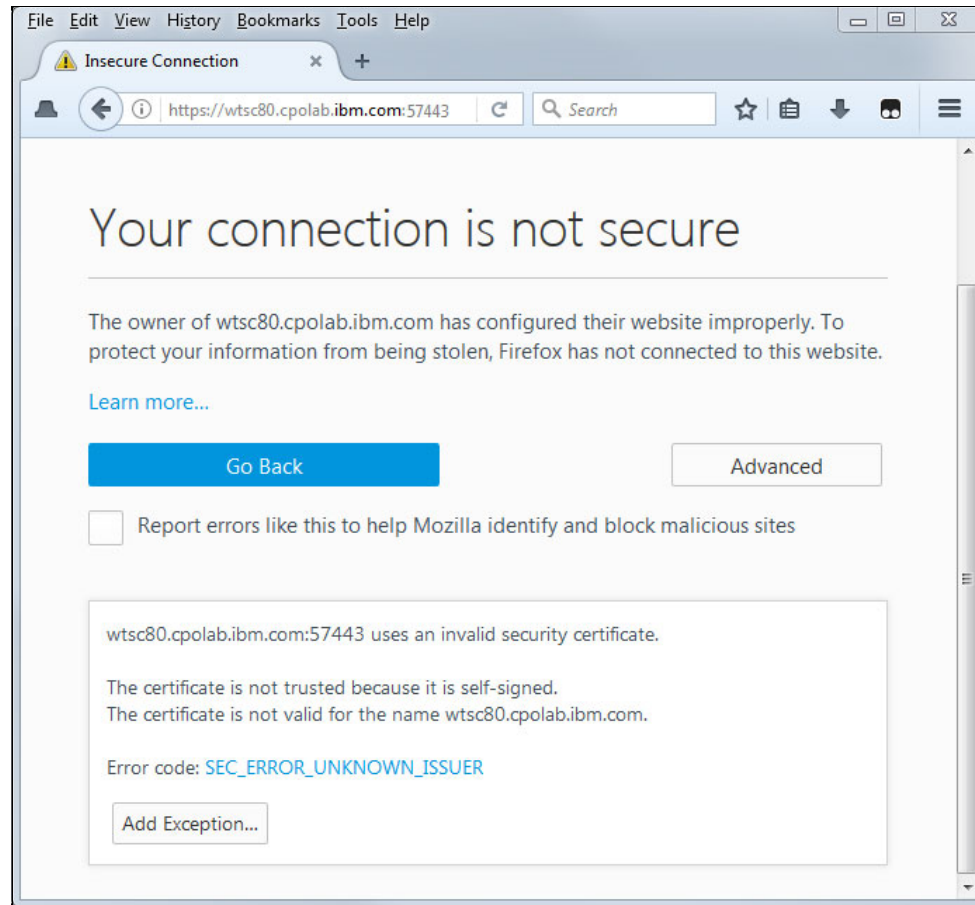


Figure 6-2 Your connection is not secure message

The error code SEC_ERROR_UNKNOWN_ISSUER clarifies the reason because the server certificate that was presented to the browser by Liberty was auto-generated by Liberty. Therefore, it is self-signed and defaults to specifying the host name as localhost in the Common Name (CN).

If you select **Add Exception**, the Add Security Exception window opens. A message indicates that the Wrong site and Unknown Identity are being used, as shown in Figure 6-3.

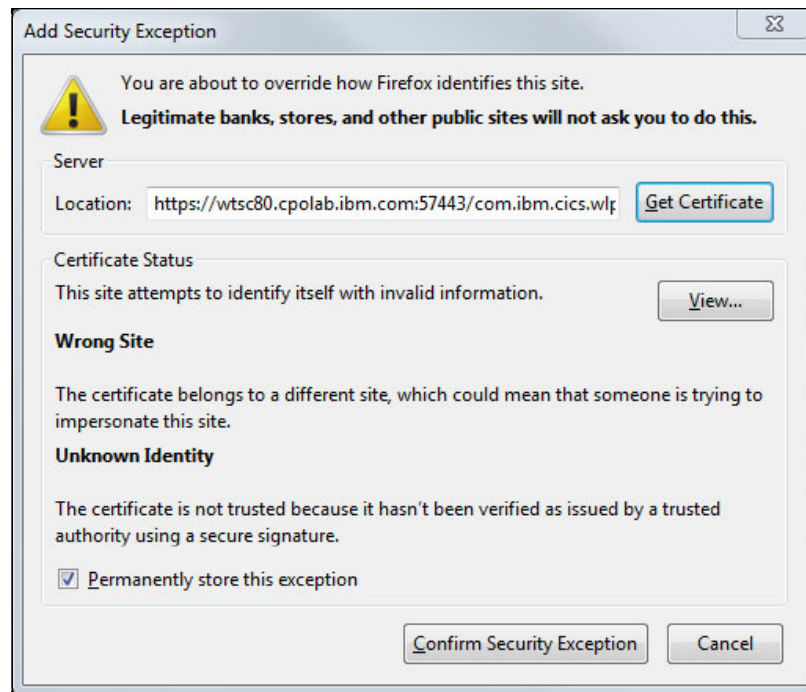


Figure 6-3 Add Security Exception message

If you click **View**, you see that the distinguished name for the Issued To and Issued By are the same, as shown in Figure 6-4. This similarity indicates that this site uses a self-signed certificate.

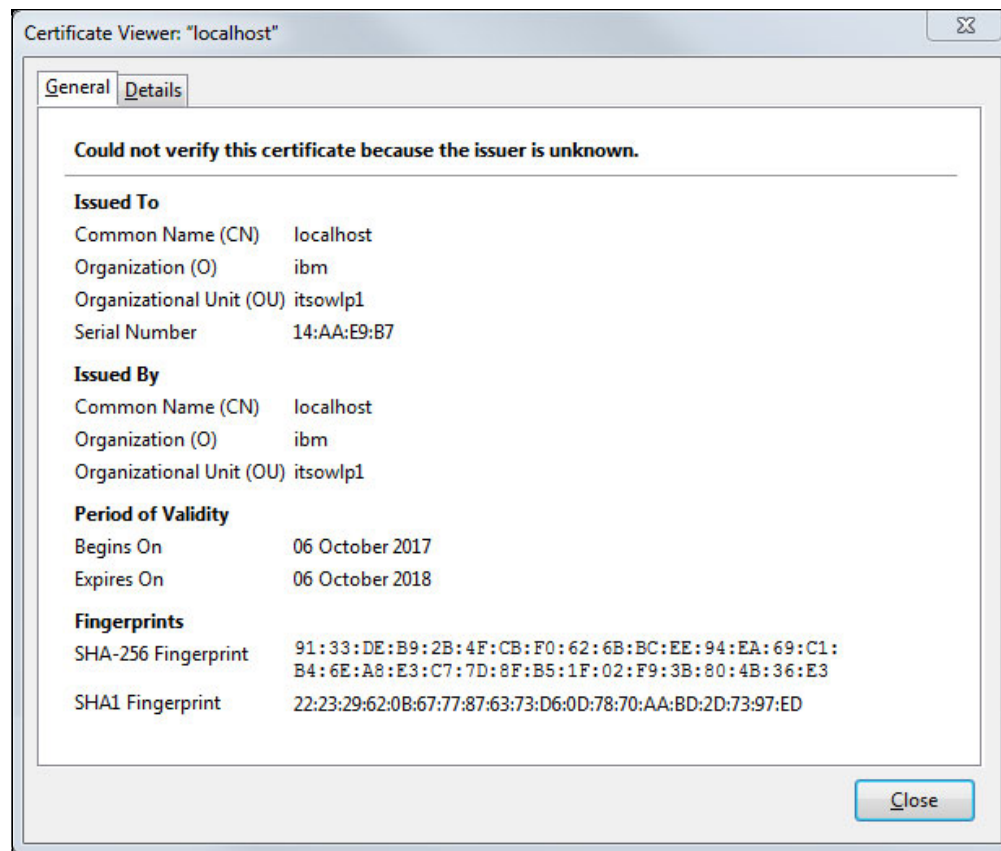


Figure 6-4 Could not verify this certificate

Also, the CN includes `localhost` as the host name. Because this name does not match the host name that is specified in the UR (in our case, `wtsc80.cpolab.ibm.com`), the Unknown Identity warning is displayed.

If you click **Confirm Security Exception** to store this exception, the HTTPS session is complete and you see the Liberty welcome page (see Figure 6-5). You now have a basic TLS setup configured. However, that setup includes a few limitations, which we describe in the following sections.

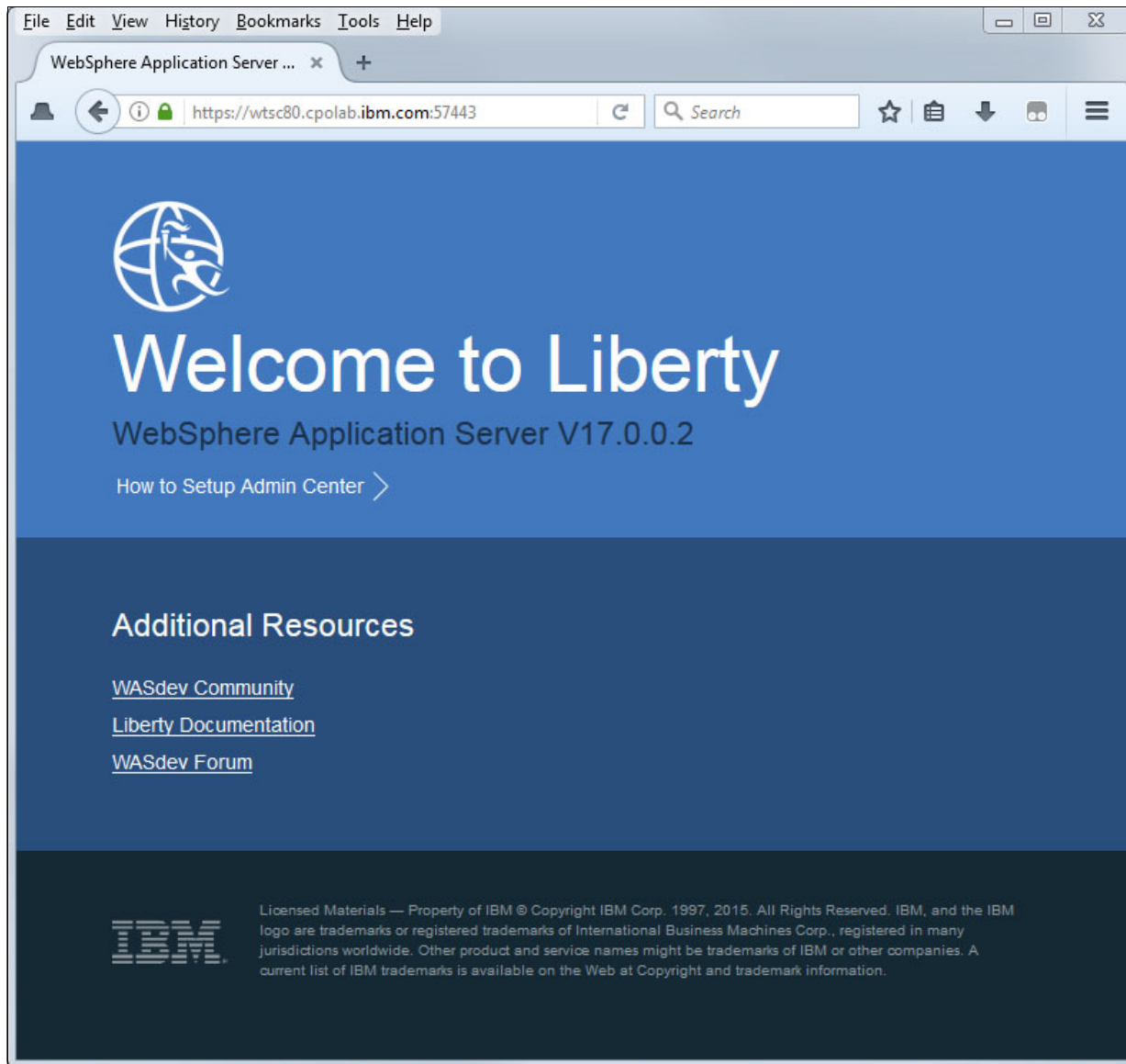


Figure 6-5 Liberty landing page that uses TLS

Note: The application that is used for testing TLS support is not important in this scenario. We used the Liberty landing page because it is available by default; however, you can use your own application, if wanted.

6.3 TLS server authentication by using a RACF key ring

The next step in our configuration process is to use RACF to store our TLS certificates, which provides the ability to integrate certificate access control with z/OS security policies and procedures. Although we used RACF in our tests, other SAF-compliant z/OS security providers can also be used.

The examples in this section use RACF to generate a self-signed certificate authority (CA) certificate. Other third-party CAs can be used and if they are used, the sequence of commands is slightly different.

Note: The user ID that is used to issue the example RACF commands in this section require the correct authority to the relevant IRR.DIGTCERT.* profiles. For more information, see the [z/OS Security Server RACF Command Language Reference](#).

Complete the following steps to configure RACF to store the TLS certificates:

1. Create a CA certificate for use as a signing certificate by using the following **RACDCERT GENCERT** command:

```
RACDCERT GENCERT CERTAUTH
      SUBJECTSDN(CN('ITSO CA') O('IBM') OU('CICS'))
      SIZE(2048) WITHLABEL('ITSO CA')
      NOTAFTER(DATE(2020-12-31))
```

2. List the CA certificate by using the following **RACF** command to verify the CA:

```
RACDCERT CERTAUTH LIST(LABEL('ITSO CA'))
```

The result of the use of the command is shown in Example 6-4.

Example 6-4 Liberty CA certificate

Digital certificate information for CERTAUTH:

```
Label: ITSO CA
Certificate ID: 2QiJmZmDhZmjgcnj4tZAw8FA
Status: TRUST
Start Date: 2017/10/09 00:00:00
End Date: 2020/12/31 23:59:59
Serial Number:
    >00<
Issuer's Name:
    >CN=ITSO CA.OU=CICS.O=IBM<
Subject's Name:
    >CN=ITSO CA.OU=CICS.O=IBM<
Signing Algorithm: sha256RSA
Key Usage: CERTSIGN
Key Type: RSA
Key Size: 2048
Private Key: YES
Ring Associations:
*** No rings associated ***
```

Note: The certificate features a Key Usage of CERTSIGN, which signifies that this certificate is a signing certificate.

3. Create a server certificate that is signed by the certificate authority from Step 1. This certificate is used to identify our Liberty server and binds the server's public key to a subject, as identified by the distinguished name (DN).

The common name in the DN should be set to the server's TCP/IP host name because this name is validated by the browser. In our case, the host name is wtsc80.cpolab.ibm.com. The label is used to identify this certificate in the RACF key store. For our certificate label, we used the concatenation of the region name and JVM server name SC8CICS7-ITSOWLP1, as shown in the following example:

```
RACDCERT ID(CICSREGN) GENCERT
  SUBJECTSDN(CN('wtsc80.cpolab.ibm.com') O('IBM') OU('CICS'))
  SIGNWITH (CERTAUTH LABEL('ITSO CA'))
  WITHLABEL('SC8CICS7-ITSOWLP1')
  NOTAFTER(DATE(2020-12-31))
  SIZE(2048)
```

4. List the contents of the new server certificate by using the following **RACF** command to verify the certificate:

```
RACDCERT ID(CICSREGN) LIST(LABEL('SC8CICS7-ITSOWLP1'))
```

The result of the use of the command is shown in Example 6-5.

Example 6-5 Liberty server certificate

Digital certificate information for user CICSREGN:

```
Label: SC8CICS7-ITSOWLP1
Certificate ID: 2QjDycPi2cXH1eLD+MPJw+L3YMnj4tbm09fx
Status: TRUST
Start Date: 2017/10/09 00:00:00
End Date: 2020/12/31 23:59:59
Serial Number:
  >03<
Issuer's Name:
  >CN=ITSO CA.OU=CICS.O=IBM<
Subject's Name:
  >CN=wtsc80.cpolab.ibm.com.OU=CICS.O=IBM<
Signing Algorithm: sha256RSA
Key Type: RSA
Key Size: 2048
Private Key: YES
Ring Associations:
  Ring Associations:
  *** No rings associated ***
```

5. Create a RACF key ring, which acts as a holder for the certificates in the RACF database. We named our key ring LIBERTY.SC8CICS7 because it is associated with our CICS region SC8CICS7 and with our CICS region user ID CICSREGN, as shown in the following example:

```
RACDCERT ID(CICSREGN) ADDRING(LIBERTY.SC8CICS7)
```


6. Connect the signing certificate and the server certificate to the RACF key ring by using the following **RACDCERT** commands:

```
RACDCERT ID(CICSREGN) CONNECT(RING(LIBERTY.SC8CICS7)
    LABEL('ITSO CA') CERTAUTH)
RACDCERT ID(CICSREGN) CONNECT(RING(LIBERTY.SC8CICS7)
    LABEL('SC8CICS7-ITSOWLP1'))
```

7. Grant the CICS region user ID the ability to read the contents of a key ring that is associated with its own user ID, as shown in the following example:

```
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY)
    ID(CICSREGN) ACCESS(READ)
```

8. Export the signing certificate to an MVS dataset to allow it to be transferred to the workstation, as shown in the following example:

```
RACDCERT CERTAUTH EXPORT(LABEL('ITSO CA'))
    DSN('CICSUSER.CERTS.ITSOCA') FORMAT(CERTDER)
```

Note: We did not specify a password on this export command because this command only exports the public key of the certificate authority, which does not warrant password protection.

9. Issue the following **RACF** command to refresh the RACF FACILITY and DIGTCERT classes:

```
SETROPTS RACLIST(FACILITY DIGTCERT) REFRESH
```

10. Transfer the signing certificate that was exported from the MVS dataset to your workstation. You should transfer this certificate in binary mode and save it with a .cert file extension.

11. Import this .cert file into Firefox by clicking in Firefox **Tools** → **Options** → **Advanced** → **Certificates** → **View Certificates** → **Import**.

12. In Firefox, click **Trust this CA to identify websites** and the new certificate authority is displayed in the Firefox Certificate Manager under Authorities as IBM → ITSO CA.

13. To use this new RACF key ring with Liberty, update your server.xml by completing the following steps:

- a. Update the location attribute in the keystore element to specify safkeyring:///LIBERTY.S8CICS7, which causes Liberty to use the CICS region user ID (CICSREGN) to access the RACF key ring LIBERTY.SC8CICS7.
- b. Update the type attribute to specify JCERACFKS, which indicates that the IBMJCE provider is used with a RACF keystore, as shown in the following example:

```
<keyStore id="defaultKeyStore"
    fileBased="false"
    location="safkeyring:///LIBERTY.S8CICS7"
    password="password"
    readOnly="true"
    type="JCERACFKS" />
```

Note: Although the password that is defined for a safkeyring keystore is not validated by RACF because access is controlled by the DIGTCERT class, it must always be specified as the string "password".

14. Restart the Liberty server and review the Liberty messages.log for the message CWWK00219I, which indicates that the HTTPS endpoint listened on port 57443.

15. Connect to the HTTPS listener that is specifying the default landing page again. You should see that the Liberty landing page (see Figure 6-5 on page 148) is secured with the TLS padlock symbol.

6.4 TLS client authentication

In this section, we describe how to configure a Liberty server to support TLS client authentication for an HTTPS connection. We used RACF to create a personal certificate that is signed by the same trust authority we used for server authentication. We then exported the client certificate and associated private key from z/OS to our workstation and imported this certificate into our web browser.

For more information about how to use `<auth-method>CLIENT-CERT</auth-method>` to map the identity from a TLS client certificate to the identity used in the Liberty server, see Chapter 7, “Securing web applications” on page 175.

Complete the following steps to configure TLS client authentication:

1. Create a personal certificate that is owned by the user ID, which is to be authenticated. Because we use the RACF user ID WEBUSER, we created this certificate by using the following RACDCERT command (see Example 6-6) that used our ITSO CA signing certificate that we created as described in 1.3, “Setting up a Liberty JVM Server” on page 11.

Example 6-6 RACDCERT GENCERT personal certificate

```
RACDCERT ID(WEBUSER) GENCERT SUBJECTSDN(CN('WINDOWSX230')
O('IBM') OU('CICS')) SIZE(2048)
SIGNWITH(CERTAUTH LABEL('ITSO CA')) WITHLABEL('WEBUSER-CERT')
```

In the CN field of the SUBJECTSDN, we specified the local host name of our test machine (WINDOWSX230).

Note: An alternative to the use of RACF to create the client certificate is to create the certificate on a third-party platform. Then, the certificate is imported into the RACF registry by using a PKCS12 file. This process can be done by using the following **RACDCERT ADD** command:

```
RACDCERT ADD('dataset') TRUST ID(userid) PASSWORD (password)
```

2. Validate the information in this new certificate by using the following command:

```
RACDCERT ID(WEBUSER) LIST( LABEL('WEBUSER-CERT'))
```

The output of the command is shown in Example 6-7.

Example 6-7 RACDCERT LIST personal certificate

Digital certificate information for user WEBUSER:

```
Label: WEBUSER-CERT
Certificate ID: 2QTXyMnT18jJ02DDxdnj
Status: TRUST
Start Date: 2017/10/09 00:00:00
End Date: 2018/10/09 23:59:59
Serial Number:
```

```

>05<
Issuer's Name:
>CN=ITSO CA.OU=CICS.O=IBM<
Subject's Name:
>CN= WINDOWSX230.OU=CICS.O=IBM<
Signing Algorithm: sha256RSA
Key Type: RSA
Key Size: 2048
Private Key: YES
Ring Associations:
*** No rings associated ***

```

3. Export the certificate in PKCS12 mode to an MVS dataset to enable it to be transferred to the workstation. Ensure that the PKCS12 package is password protected because it now contains the private key, as shown in the following example:

```

RACDCERT ID(WEBUSER) EXPORT(LABEL('WEBUSER-CERT'))
DSN('CICSUSER.CERTS.WEBUSER')
FORMAT(PKCS12DER) PASSWORD('ITSO')

```

Note: We used the PKCS12 file format because this format exports the certificate and the private key, which are required if this format is to be used as a client certificate.

4. Transfer to the workstation in binary mode by using FTP and save as a file with a .p12 extension.
5. Import the .p12 file into your web browser by clicking (in Firefox) **Tools** → **Options** → **Advanced** → **Certificates** → **View Certificate** → **Your Certificates**.
6. Click **Import** and then, browse to the location of your .p12 file. You are presented with a Password Required window in which you enter the password that is used to encrypt the certificate. Enter the password value that was used in Step 3.

If this process is successful, you see that the certificate is installed, as shown in Figure 6-6.

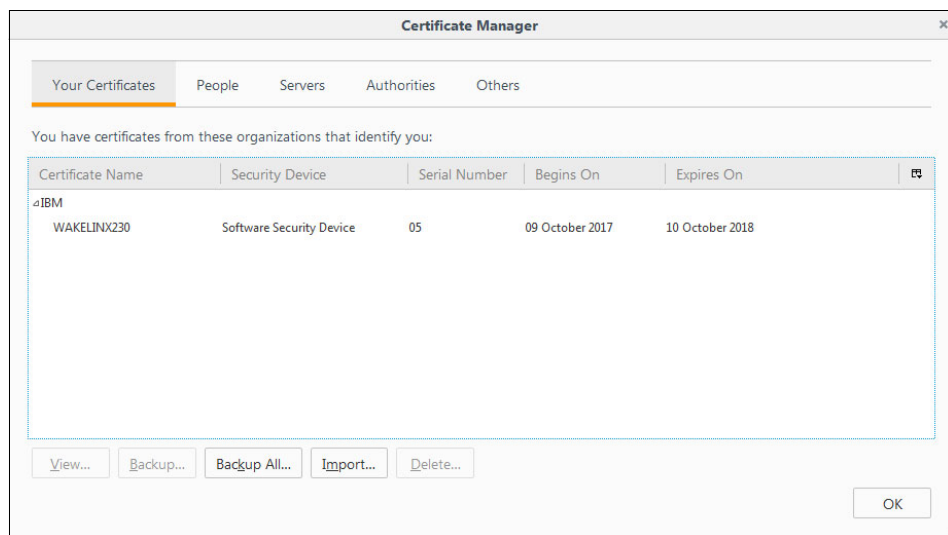


Figure 6-6 Certificate Manager window

7. To enforce client authentication during the TLS handshake, we now must change the Liberty configuration in the `server.xml`. This process requires that the `clientAuthentication` attribute in the `ssl` element is changed to `true`. The changes to the `ssl` element in our `server.xml` is shown in Example 6-8.

Example 6-8 Server.xml: clientAuthentication attribute

```
<ssl id="defaultSSLConfig"
      clientAuthentication="true"
      keyStoreRef="defaultKeyStore"
      serverKeyAlias="SC8CICS7-ITSOWLP1" />
```

8. Start the JVM server and connect to the HTTPS listener that specifies the Liberty landing page again. You are prompted to choose a client certificate to present as identification, as shown in Figure 6-7.

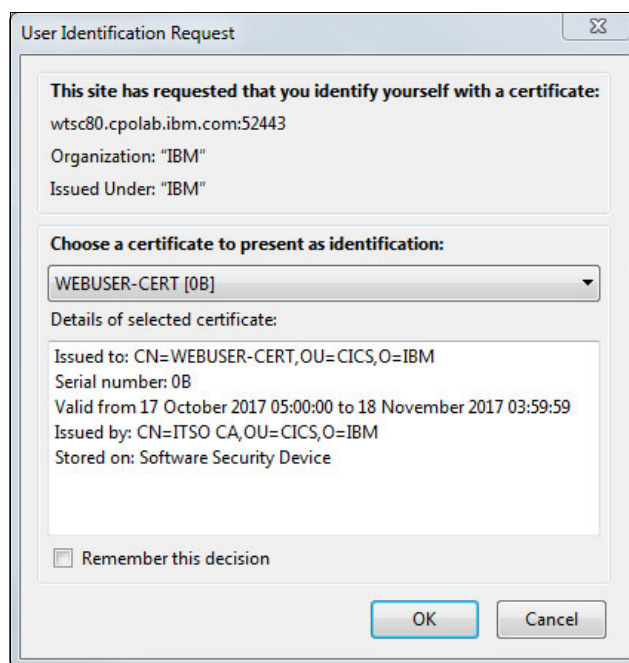


Figure 6-7 User Identification Request window

9. Select the certificate that you imported and click **OK**. The TLS handshake should complete and the Liberty welcome page is displayed with the lock symbol, as shown in Figure 6-5 on page 148.

Making client authentication optional

If `clientAuthentication="true"` but the client does not include a certificate or the certificate is not trusted by the server, the TLS handshake and HTTP connection fail. If you do not want to cause a failure in this scenario, you can specify the attribute `clientAuthenticationSupported="true"` instead of `clientAuthentication="true"`. This change causes the server to request a certificate from the client, but allow the connection to proceed without one.

Note: If client certificate authentication is specified in the application's `web.xml` by using `<auth-method>CLIENT-CERT</auth-method>` and no TLS client certificate is available, Liberty provides the following `server.xml` attribute:

```
<webAppSecurity allowFailOverToBasicAuth="true" />
```

If specified, this attribute causes Liberty to fail over to basic authentication for any application in which client certificate authentication fails.

Adding a truststore

When client authentication is configured, Liberty asks the client to provide its certificate for each new HTTPS connection. Then, it validates the chain of trust for that certificate by using its truststore. That is, it validates that the client certificate issuer is in the RACF key ring.

The server does not need access to the client certificate, only the signing certificate. In our configuration, the truststore defaults to the keystore. However, if a separate signing certificate is used to sign the client certificate, it can be added to a specific RACF key ring by using the following commands (assuming that the certificate label for the client's signing certificate is `CLIENT CA`):

```
RACDCERT ID(CICSREGN) ADDRING(LIBERTY.TRUST.SC8CICS7)
RACDCERT ID(CICSREGN) CONNECT(RING(LIBERTY.TRUST.SC8CICS7)
LABEL('CLIENT CA') CERTAUTH)
```

This key ring should then be specified in a `trustStoreRef` element in `server.xml` that refers to another keystore that references the separate key ring, as shown in Example 6-9.

Example 6-9 Server.xml: trustStoreRef for client CA

```
<ssl id="defaultSSLConfig"
  clientAuthentication="true"
  keyStoreRef="defaultKeyStore"
  trustStoreRef="trustStore"
  serverKeyAlias="SC8CICS7-ITSOWLP1" />
<keyStore id="defaultKeyStore"
  fileBased="false"
  location="safkeyring:///LIBERTY.S8CICS7"
  password="password"
  readOnly="true"
  type="JCERACFKS" />
<keyStore id="trustStore"
  fileBased="false"
  location="safkeyring:///LIBERTY.TRUST.S8CICS7"
  password="password"
  readOnly="true"
  type="JCERACFKS" />
```

6.5 Hints and tips when using TLS

In this section, we describe how to analyze and optimize the TLS support that is used in Liberty.

6.5.1 Tracing TLS

Several methods are available for tracing the usage of TLS in Liberty. In this section, we describe some of the techniques that we used.

JSSE tracing

JSSE-specific debug tracing support is controlled by using the system property `javax.net.debug`. This support traces different components of the JSSE SSL implementation. For more information, see [IBM Knowledge Center](#).

To enable JSSE tracing, we set the following system property in our CICS JVM profile:

```
-Djavax.net.debug=ssl
```

This property causes JSSE tracing to be written to the JVM server stdout, which can also be viewed in the Liberty messages.log output.

With tracing enabled at startup of the JVM, you see the output that is shown in Example 6-10 when the IBM JSSE provider initializes, which lists the installed JCE providers.

Example 6-10 JSSE trace initialization

```
IBMJSSEProvider2 Build-Level: -20170728
Installed Providers =
  IBMJCECCA
  IBMJCE
  IBMJGSSProvider
  IBMJSSE2
  IBM CertPath
  IBMSASL
  IBMXMLCRYPTO
  IBMXMLEnc
  IBMSPNEGO
  SUN
```

```
.....
```

Liberty SSL tracing

The Liberty logging component can be controlled by using the `server.xml` configuration. We used the `logging` element as shown in Example 6-11 to trace the `SSLChannel` and `SSL` components in our Liberty server. The output was written to the specified `ssl_trace.log` in the default `${server.output.dir}/logs` directory. The trace is useful to validate if errors in the Liberty SSL configuration existed.

Example 6-11 server.xml: traceSpecification

```
<logging traceSpecification="SSLChannel=all:SSL=all"
  traceFileName="ssl_trace.log"
  maxFileSize="20"
  maxFiles="10"
  traceFormat="BASIC" />
```

6.5.2 Enforcing TLS for web applications

The following options are available to enforce that TLS is used for HTTP connections into Liberty:

- ▶ Disable the HTTP endpoint
- ▶ Use a security constraint in the `web.xml`

These options are described next.

Disabling the HTTP endpoint

The simplest option to enforce that TLS is used is to disable the use of the Liberty HTTP endpoint. This endpoint can be disabled by updating the `httpPort` attribute to specify `httpPort="-1"`, as shown in Example 6-12.

Example 6-12 server.xml: Enforcing TLS

```
<httpEndpoint id="defaultHttpEndpoint"
  host="wtsc80"
  httpOptionsRef="httpoptions"
  httpPort="-1"
  httpsPort="57443" />
```

Warning: If the `httpPort` attribute is removed from the `httpEndpoint`, the default value of 9080 is assumed and the server starts to listen on this port.

Security constraint

Another mechanism to enforce TLS usage is create a rule in each application by adding the following `transport-guarantee` element to the security constraint in the application's `web.xml`:

```
<transport-guarantee>CONFIDENTIAL</transport-guarantee>
```

If you specify `CONFIDENTIAL` as the value, all requests that match the constraint pattern are automatically redirected to the HTTPS endpoint.

We added a `web.xml` to our `restapp` sample (see Example 6-13 on page 158) and redeployed this version of our application into our Liberty server.

Example 6-13 Web.xml: Enforcing TLS

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>restapp</web-resource-name>
    <url-pattern>/com.ibm.cicsdev.restapp/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

We then connected our browser to the insecure cicsinfo service at the following URL:

`http://wtsc80:57080/com.ibm.cicsdev.restapp/rest/cicsinfo`

Our browser was then redirected to the secure HTTPS listener on port 57443.

6.5.3 HTTP persistent connections

The most intensive encryption process when TLS is the RSA or Elliptic Curve Cryptography (ECC) that is used during the handshake phase to encrypt and sign messages. It also produces the symmetric session keys that are used for payload encryption.

The simplest mechanism to reduce this cost is to enable persistent HTTP connections because a TLS handshake occurs only during the creation of the HTTP connection. By using this mechanism, the cost of the handshake is spread over multiple requests.

In Liberty, the default number of persistent requests that can reuse an HTTP connection is 100. After 100 requests, the HTTP connection and the underlying socket are closed. If HTTPS is used, a full TLS handshake occurs every 100 requests. To improve performance, you can increase this value by using the `maxKeepAliveRequests` attribute on an `httpOptions` element that is referenced from the `httpEndpoint`, as shown in Example 6-14.

Example 6-14 httpOptions element

```
<httpEndpoint id="defaultHttpEndpoint"
  host="wtsc80"
  httpOptionsRef="httpoptions"
  httpPort="57080"
  httpsPort="57443" />
<httpOptions id="httpoptions"
  maxKeepAliveRequests="10000" />
```

The setting for `maxKeepAliveRequests` is important because not only can this setting affect performance, it affects workload distribution in a high availability (HA) cluster. For more information about how we configured HA with Liberty, see Chapter 9, “Port sharing and cloning regions” on page 253.

Note: To further improve the performance of TLS handshaking, the IBMJCECCA provider can also be used. This provider uses the ICSF callable service to drive the IBM Crypto Express cards. For more information, see 6.6, “Using cryptographic hardware with JSSE” on page 164.

TCP/IP netstat

Liberty does not monitor HTTP persistent session usage. However, the **TCP/IP NETSTAT** command can be used to query the number of socket connection requests when a TCP/IP listening port is used.

In our configuration, we ran a workload of 100 requests by using a curl script. Then, we issued the following **MVS** command, which returned the information that is shown in Example 6-15:

```
/D TCPIP,,NETSTAT,ALL,PORT=57443
```

Example 6-15 NETSTAT display

```
EZZ2500I NETSTAT CS V2R3 TCPIP 584
CLIENT NAME: SC8CICS7                CLIENT ID: 00044C1A
LCLSOCK: 9.76.61.131..57443          FGNSOCK: 0.0.0.0..0
  BYTESIN: 0000000000                BYTESOUT: 0000000000
  SEGMENTSIN: 0000000000            SEGMENTSOUT: 0000000000
  STARTDATE: 10/18/2017              STARTTIME: 13:00:15
  LAST TOUCHED: 13:01:02              STATE: LISTEN
  RCVNXT: 0000000000                 SNDNXT: 0000000000
  CLIENTRCVNXT: 0000000000           CLIENTSNDNXT: 0000000000
  INITRCVSEQNUM: 0000000000          INITSNDSEQNUM: 0000000000
  CONGESTIONWINDOW: 0000000000       SLOWSTARTTHRESHOLD: 0000000000
  INCOMINGWINDOWNUM: 0000000000      OUTGOINGWINDOWNUM: 0000000000
  SNDWL1: 0000000000                 SNDWL2: 0000000000
  SNDWND: 0000000000                 MAXSNDWND: 0000000000
  SNDUNA: 0000000000                 RTT_SEQ: 0000000000
  MAXIMUMSEGMENTSIZE: 0000000536     DSFIELD: 00
  ROUND-TRIP INFORMATION:
    SMOOTH TRIP TIME: 0.000           SMOOTHTRIPVARIANCE: 1500.000
  REXMT: 0000000000                  REXMTCOUNT: 0000000000
  DUPACKS: 0000000000                RCVWND: 0000032768
  SOCKOPT: 80                        TCPTIMER: 00
  TCPSIG: 00                         TCPSEL: 00
  TCPDET: C0                         TCPPOL: 08
  TCPPRF: 00                         TCPPRF2: 00
  TCPPRF3: 00
  QOSPOLICY: NO
  TTLSPOLICY: NO
  ROUTINGPOLICY: NO
  RECEIVEBUFFERSIZE: 0000016384      SENDBUFFERSIZE: 0000016384
  CONNECTIONSIN: 0000000100          CONNECTIONSDROPPED: 0000000000
  MAXIMUMBACKLOG: 0000000010         CONNECTIONFLOOD: NO
  CURRENTBACKLOG: 0000000000
  SERVERBACKLOG: 0000000000          FRCABACKLOG: 0000000000
  CURRENTCONNECTIONS: 0000000000     SEF: 100
  QUIESCED: NO
```

The value of the **CONNECTIONSIN** is 100, which indicates that our workload did not use persistent connections because 100 socket connections were created for the 100 requests.

Note: If the **NETSTAT** command returns multiple records, you should use the record that specifies **STATE:LISTEN** because this record is for the listening socket, rather than the ephemeral sockets that were established for the browser sessions.

6.5.4 TLS session timeout

When HTTP connections timeout, the effect of full handshakes can be avoided by reusing the SSL session ID. Reusing the session ID is known as a *null handshake*, and it is considerably cheaper to perform. The time interval in the server during which the server allows session IDs to be resumed is configured by setting the `sslSessionTimeout` attribute on the `sslOptions` element. The default setting is 8640 ms and can be extended to 10 minutes, as shown in the following example:

```
<sslOptions id="mySSLOptions"
  sslRef="DefaultSSLSettings"
  sslSessionTimeout="10m" />
```

Increasing the session timeout can be a performance benefit if HTTP connections are not long-lived, and TLS clients must reconnect frequently. For optimum performance, the `sslSessionTimeout` attribute must be set to longer than the time between client HTTP requests. Setting this attribute allows the server to reuse session IDs.

6.5.5 Controlling the TLS version

To control the TLS version that is supported by the server, the `sslProtocol` attribute on the `ssl` element is updated in `server.xml` to ensure that only a specific version is used. We found protocol versions can be specified in the `sslProtocol` attribute, which results in the SSL/TLS versions being supported on the SSL/TLS handshake, as listed in Table 6-1.

Table 6-1 SSL/TLS protocol versions in JSSE

sslProtocol attribute	Supported protocol versions			
	SSL V3.0	TLS V1.0	TLS V1.1	TLS V1.2
SSL	No *	Yes	Yes	Yes
SSLv3	No *	No	No	No
TLS	No	Yes	No	No
TLSv1	No	Yes	No	No
TLSv1.1	No	No	Yes	No
TLSv1.2	No	No	No	Yes
SSL_TLS	No *	Yes	No	No
SSL_TLSv2	No *	Yes	Yes	Yes

Note: * SSL v3 and v2 is disabled by default in IBMJSSE and in many browsers because of security vulnerabilities. Its use is no longer recommended.

The latest version of TLS is TLS v1.2 and provides for the most secure set of ciphers. We made the change that is shown in Example 6-16 on page 161 to our `server.xml` to enforce use of TLSv1.2 in our server. For more information about SSL protocols that are supported in JSSE, see the [Protocols section of the IBM SDK, Java Technology Edition 8.0.0 page](#) of IBM Knowledge Center.

```
<ssl id="defaultSSLConfig"
    keyStoreRef="defaultKeyStore"
    sslProtocol="TLSv1.2" />
```

If the client and server do not have a mutually acceptable cipher suite, you might see the message `SSL_ERROR_NO_CYPHER_OVERLAP` returned by your web browser, as shown in Figure 6-8. This message can be caused by a too restrictive set of cipher suites being specified, or that the client might not support the same TLS version as the server.

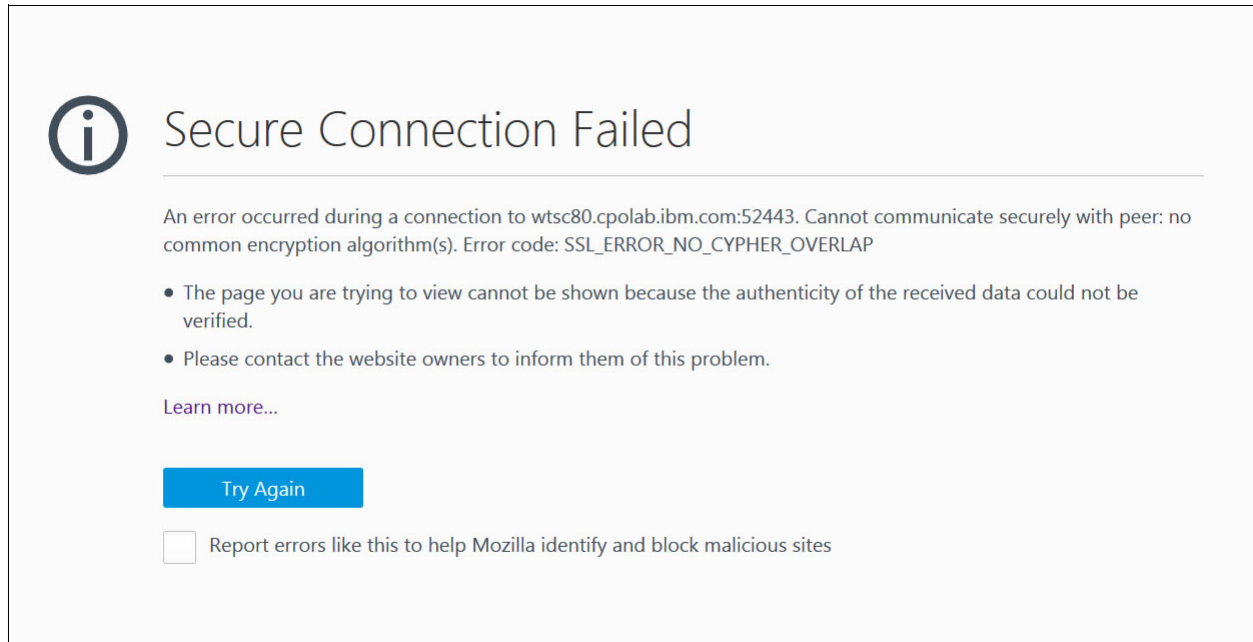


Figure 6-8 *Secure Connection Failed message*

In this situation, you might need to modify the `sslProtocol` attribute in `server.xml` to specify a less restrictive range of TLS versions. You also can allow a less restrictive range of cipher suites to be used (for more information, see 6.5.6, “Controlling the cipher suite” on page 161).

6.5.6 Controlling the cipher suite

The process in which the cipher that is used for a TLS connection is negotiated during the TLS handshake includes the following steps:

1. The TLS client sends a “client hello” message that lists cryptographic information, including the SSL or TLS version and, in the client’s order of preference, the cipher suites that are supported by the client.
2. TLS server responds with a “server hello” message that contains a mutually agreeable cipher suite that is chosen by the server from the list that is provided by the client.

Different TLS ciphers feature different encryption strengths and performance characteristics. Therefore, it is often useful to understand which cipher suite was negotiated between the client and the server.

Most browsers can display the negotiated cipher suite. You can determine which cipher suite was negotiated on the server by using JSSE tracing, as described in 6.5.1, “Tracing TLS ” on page 156.

By using the JSSE trace log, you can search for the string `%% Negotiating`. In our example, we saw that the following cipher suite was chosen during the negotiation between the browser and the Liberty server:

```
%% Negotiating: [Session-100, SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256]
```

This cipher suite defines the use of the following encryption algorithms:

- ▶ Key exchange: Ephemeral Elliptic Curve Diffie-Hellman with RSA (ECDHE_RSA)
- ▶ Block cipher: AES_GCM encryption algorithm with a 128-bit key (AES_128_GCM)
- ▶ Data integrity: SHA hash algorithm with a 256-bit key (SHA256)

To modify the list of ciphers that are supported by the server, they can be added as a white space-separated list in the `enabledCiphers` attribute on the `ssl` element. We modified our `server.xml` as shown in Example 6-17 to support the following ciphers:

- ▶ `SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- ▶ `SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256`

Example 6-17 Liberty server.xml: Enabled ciphers

```
<ssl id="defaultSSLConfig"
    enabledCiphers="SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256
        SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256"
    keyStoreRef="defaultKeyStore"
    sslProtocol="TLSv1.2" />
```

For more information about the supported ciphers in the IBM Java z/OS SDK, see [the Cipher Suites topic of the IBM SDK, Java Technology Edition 8.0.0 page](#) of IBM Knowledge Center.

6.5.7 Restricting weak algorithms

One useful customization feature of JSSE is the ability to restrict ciphers and algorithms to a specific minimum strength. This restriction can be achieved by using the `jdk.tls.disabledAlgorithms` property, which can be specified in the Java security policy.

By default, the IBM Java SDK disables specific algorithms, such as RSA key sizes less than 1024 bits. We restricted this behavior further to RSA key sizes less than 2048 bits as shown in the following process:

1. We copied the `java.security` file from `$JAVA_HOME/lib/security` to `/var/cics/ibmjce/java.security.2048` and set the location by using the following `java.security.properties` system property in our JVM server profile:
`-Djava.security.properties=/var/cicsts/ibmjce/java.security.2048`
2. We modified the `jdk.tls.disabledAlgorithms` property in this file to restrict RSA key sizes to 2048 bits or greater as shown in the following example:
`jdk.tls.disabledAlgorithms= RSA keySize < 2048, SSLv3, RC4, MD5withRSA, DH keySize < 768, 3DES_EDE_CBC, DESede, EC keySize < 224`
3. We followed the processing (as described in 6.4, “TLS client authentication” on page 152) to generate a personal certificate in RACF. However, this time with a key size of 1024 (see Example 6-18 on page 163).

Example 6-18 RACDCERT GENCERT: Weak personal certificate

```
RACDCERT ID(WEBUSER) GENCERT
  SUBJECTSDN(CN('WINDOWSX230') O('IBM') OU('CICS'))
  SIZE(1024) SIGNWITH(CERTAUTH LABEL('ITSO CA'))
  WITHLABEL('WEAK-CERT')
```

4. We exported this personal certificate as a PKSC12 file and transferred this file to our workstation. Then, we imported this file into our Mozilla web browser by using the same procedure as described in 6.4, “TLS client authentication” on page 152.
5. We opened a new private tab in our Mozilla browser and connected to the Liberty HTTPS port. Then, we specified the default Liberty landing page as `https://wtsc80:57443`.

We were asked to choose a client certificate to present as identification, as shown in Figure 6-9.

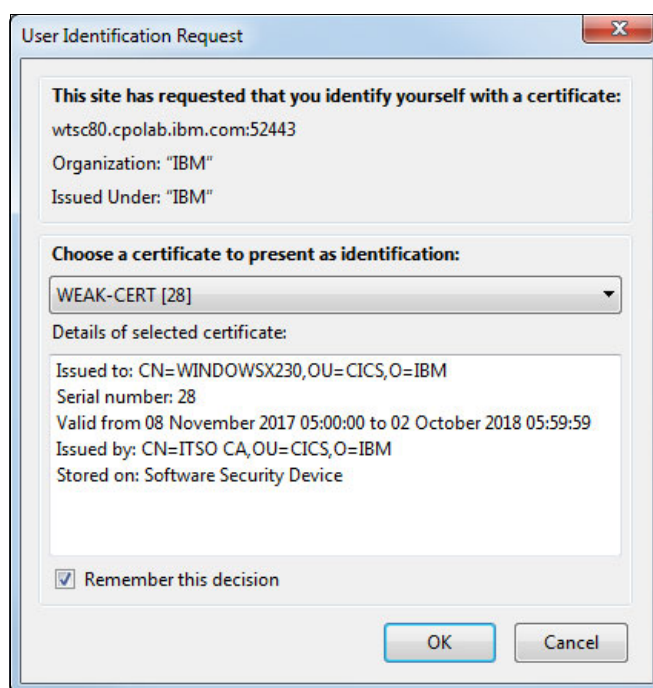


Figure 6-9 User Identification Request window

6. We selected the certificate that was labeled **WEAK-CERT** and clicked **OK**.
7. The browser reported the error Secure Connection Failed.

By using `javax.net.debug` tracing as described in 6.5.1, “Tracing TLS” on page 156, we also found the following exception in the JSSE trace that explains that the TLS connection failed because the certificate that was presented did not match our rule of `RSA keySize < 2048`:

```
java.security.cert.CertificateException: Certificates does not conform to
algorithm constraints
```

6.6 Using cryptographic hardware with JSSE

In this section, we describe how to use the IBM Z cryptographic hardware with JSSE.

6.6.1 Cryptographic hardware

Two cryptographic hardware devices are available on IBM Z: CP Assist for Cryptographic Function (CPACF) and IBM Crypto Express cards, which are supported in different ways by the two IBMJCE providers we are using, as shown in Figure 6-10.

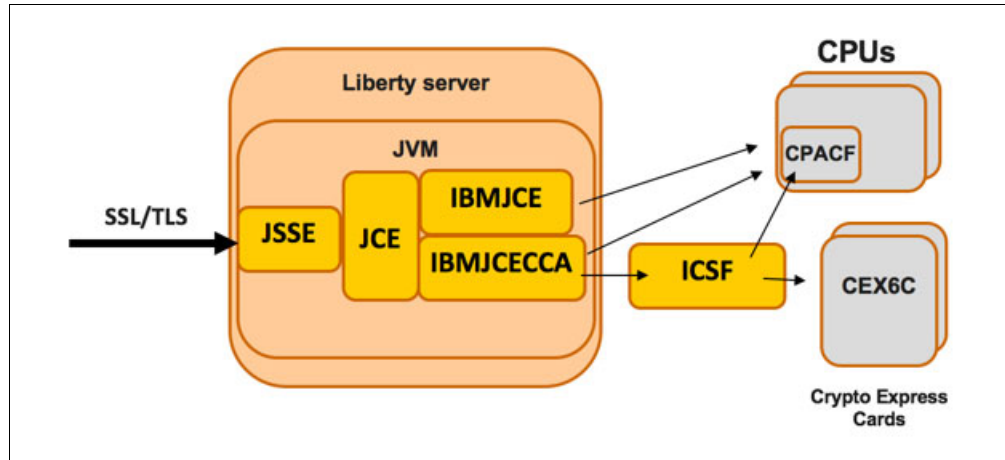


Figure 6-10 JSSE and IBM Z cryptographic hardware

CP Assist for Cryptographic Function

CPACF is a set of cryptographic instructions that are available on all CPs, including zIIPs, IFLs, and General Purpose CPUs. The CPACF is accessible through native assembler instructions (such as the KM cipher message), or by using the callable services that are available with ICSF. It provides symmetric key cryptography for clear key and protected key scenarios, along with support for hashing functions.

Various symmetric algorithms are supported by the CPACF, including DES, 3DES, and AES-CBC and AES-GCM. CPACF also supports random number generation and SHA-based digest algorithms.

Use of the CPACF provides the potential for significantly improved performance for symmetric encryption with clear keys, which is the encryption mechanism that is used for block and stream ciphers in the TLS protocol.

To ascertain whether the CPACF is enabled on your system, check the ICSF startup log for the production of the following message:

```
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE
```

IBM Crypto Express cards

The IBM Crypto Express cards are optional I/O attached cards that implement more cryptographic functions. On an IBM z14, this feature is available as a Crypto Express 6S (CEX6S) adapter, or Crypto Express 5S (CEX5S).

The CEX6S adapter can be configured in one of the following modes:

- ▶ CEX6C: Common Cryptographic Architecture (CCA) coprocessor

By default, this card is a coprocessor and can support a wider range of callable services, including secure key and clear key support for PKA decrypt, digital signature verify, and digital signature generate (including RSA and ECC variants).

- ▶ CEX6A: Common Cryptographic Architecture accelerator

This card can be configured as an accelerator. In this mode, the card supports only three clear key cryptographic APIs that are associated with RSA public key encryption, decryption, and verification.

When the cryptographic coprocessor is configured as an accelerator, it provides better throughput at the expense of supporting fewer services.

- ▶ CEX6P: IBM Enterprise Public-Key Cryptography Standards (PKCS) #11 (EP11) coprocessor

Support is available for EP11 or PKCS #11 Enterprise mode. In this mode, the card supports only APIs associated with PKCS #11.

Note: At least one Crypto Express card that is configured as a coprocessor is required for IBMJCECCA to initialize. To remove the coprocessor as a single point of failure, it is common for two Crypto Express cards to be configured in coprocessor mode when supporting TLS usage with JSSE.

For more information, see the IBM Techdoc [A Synopsis of z Systems Crypto Hardware](#), WP100810.

6.6.2 Cryptographic software

In this section, we describe the different software components that are necessary to use the cryptographic hardware with Java on z/OS.

Integrated Cryptographic Service Facility

The Integrated Cryptographic Service Facility (ICSF) works with the hardware cryptographic features and RACF to provide secure, high-speed cryptographic services in the z/OS environment. As new cryptographic functions are implemented in the hardware, new versions of ICSF are made available to start those functions. ICSF is available as a component of and packaged with z/OS; however, the most current versions are available at [the IBM z/Os downloads page](#) of the IBM IT infrastructure website.

For more information about hardware support for the various versions of ICSF, see the Techdoc [z/OS: ICSF Version and FMID Cross Reference](#), TD103782.

Note: The CSFSERV class controls access to ICSF callable services. This class was *not* active in our testing. If it is active, access must be granted when generating keys or calling ICSF by using the IBMJCECCA provider.

IBMJCE

The IBM JVM default security provider is IBMJCE, which is the default implementation of the Java Cryptographic Extension (JCE) on z/OS. When Java 8 is used, the IBMJCE provider does not use the services of ICSF; instead, it uses native assembler instructions to detect and use the CPACF. This support for CPACF provides improved performance for symmetric key encryption (for example, AES and 3DES) and hashing algorithms (for example, SHA1 and SHA2) that are used in bulk encryption with TLS because the ability to avoid JNI calls to ICSF and thus improves potential zIIP offload.

For more information about performance with Java 8 on IBM Z and IBMJCE, see the [IBM Z Development Blog and Mainframe Insights](#) blog post.

IBMJCECCA

The IBM Java SDK on z/OS provides another security provider that is named IBMJCECCA, which is integrated with ICSF and can use the Crypto Express cards to assist with asymmetric key encryption. Similar to the IBMJCE provider, the IBMJCECCA provider also uses native assembler instructions to directly call the CPACF to assist with specific symmetric key encryption algorithms.

The following key facilities are provided by IBMJCECCA:

- ▶ Digital signing and verify functions, which are used during public key exchange for TLS.
- ▶ Secure and protected key support (that is, keys that are never available in the clear).

To benefit from the Crypto Express card, the IBMJCECCA provider must be added to the JCE providers list at a higher position than the IBMJCE provider. ICSF must also be started.

Use of IBMJCECCA can provide a performance advantage over IBMJCE when performing asymmetric encryption during the secret key exchange that is involved in the TLS handshaking phase.

Note: For IBMJCECCA to initialize, ICSF must be started and at least one coprocessor must be available. We found in our testing that if IBMJCECCA cannot be loaded at initialization, the IBMJCE provider is loaded instead because this provider was the next provider in our java security provider list.

IBMJCEHYBRID

The IBM JCE hybrid provider is designed to enable an application to use the underlying JCE providers without being concerned whether the hardware cryptographic features are available. If the IBMJCEHYBRID provider is used, the HTTPS support can continue to function, even if the underlying hardware or ICSF is not available.

Configuring the JCE providers

The different JCE providers that are available and how to configure their usage in the server.xml configuration file are listed in Table 6-2.

Table 6-2 JCE provider and server.xml configuration settings

JCE provider	server.xml	
	ssl keystore location attribute	ssl keystore type attribute
IBMJCE	location="safkeyring:///"	type="JCERACFKS"
IBMJCECCA	location="safkeyringhw:///"	type="JCECCARACFKS"
IBMJCEHYBRID	location="safkeyringhybrid:///"	type="JCEHYBRIDRACFKS"

For more information about configuring the JCE providers on z/OS, see the [z/OS Java Security Frequently Asked Questions](#).

6.6.3 Configuring TLS to use the cryptographic coprocessors

In this section, we describe how to configure IBMJCECCA to use the cryptographic coprocessors on an IBM z14.

To use the IBMJCECCA provider on our system, we performed the following configuration steps:

1. We [downloaded and installed](#) the latest version of ICSF to obtain the updates for z/OS 2.3 support.
2. We copied the security provider file `java.security` from `$JAVA_HOME/lib/security` to the location `/var/cicsts/ibmjce/java.security.jcecca` and added IBMJCECCA to the security provider list, as shown in Example 6-19.

Example 6-19 IBMJCECCA in the security provider list.

```
# Java security providers
security.provider.1=com.ibm.crypto.hdwCCA.provider.IBMJCECCA
security.provider.2=com.ibm.crypto.provider.IBMJCE
security.provider.3=com.ibm.jsse2.IBMJSSEProvider2
security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
security.provider.6=com.ibm.security.sasl.IBMSASL
security.provider.7=com.ibm.xml.crypto.IBMXMLCryptoProvider
security.provider.8=com.ibm.xml.enc.IBMXMLEncProvider
security.provider.9=com.ibm.security.jgss.mech.spnego.IBMSPNEGO
security.provider.10=sun.security.provider.Sun
```

3. We updated our JVM to use this new provider list by adding the following system property to our JVM server profile:
`-Djava.security.properties=/var/cicsts/ibmjce/java.security.jcecca`
4. We updated our `server.xml` to use the `safkeyringhw` prefix in the `location` attribute, and set the `JCECCARACFKS` type on the `keyStore` element, as shown in Example 6-20.

Example 6-20 Liberty server.xml - JCECCARACFKS settings

```
<keyStore id="racfKeyStore"
  fileBased="false"
  location="safkeyringhw:///LIBERTY.SC8CICS7"
  password="password"
  readOnly="true"
  type="JCECCARACFKS" />
```

5. We restarted our CICS JVM server, established an HTTPS connection to our Liberty default application, and verified that the web page successfully loaded.

IBMJCEHYBRID

The IBMJCEHYBRID provider is designed to use cryptographic hardware and processors when they are available, but continues without those cryptographic features when they are not available.

To use IBMJCECCA provider, we performed the following steps after configuring IBMJCECCA:

1. We added the IBMJCEHYBRID provider to the top of our Java security provider list in `/var/cicsts/ibmjce/java.security.jcecca`, as shown in Example 6-21.

Example 6-21 IBMJCEHYBRID in the security provider

```
security.provider.1=com.ibm.crypto.ibmjcehybrid.provider.IBMJCEHYBRID
security.provider.2=com.ibm.crypto.hwcca.provider.IBMJCECCA
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.jsse2.IBMJSSEProvider2
security.provider.5=com.ibm.security.jgss.IBMJGSSProvider
security.provider.6=com.ibm.security.cert.IBMCertPath
security.provider.7=com.ibm.security.sasl.IBMSASL
security.provider.8=com.ibm.xml.crypto.IBMXMLCryptoProvider
security.provider.9=com.ibm.xml.enc.IBMXMLEncProvider
security.provider.10=com.ibm.security.jgss.mech.spnego.IBMSPNEGO
security.provider.11=sun.security.provider.Sun
```

2. We updated our `server.xml` to use the `safkeyringhybrid` keyword and the `JCEHYBRIDRACFKS` type, as shown in Example 6-22.

Example 6-22 Liberty server.xml

```
<keyStore id="racfKeyStore"
  fileBased="false"
  location="safkeyringhybrid:///LIBERTY.SC8CICS7"
  password="password"
  readOnly="true"
  type="JCEHYBRIDRACFKS" />
```

3. We restarted our CICS JVM server, established an HTTPS connection, and verified the web page successfully loaded.

Note: In our testing, we found that if no cryptographic coprocessors were available when the JVM server started or ICSF was not started, the IBMJCEHYBRID provider silently failed over to use IBMJCE. However, if the cryptographic coprocessor went offline while the JVM was running, the JVM server needed to be restarted to allow TLS requests to continue to be successfully processed because calls to the service CSNDPKI (PKA key import) failed, which resulted in runtime Exceptions.

6.6.4 Monitoring cryptographic hardware

The IBMJCE and the IBMJCECCA provider can use cryptographic hardware. However, one of the advantages of using IBMJCECCA is that various tools are available to monitor the use of the cryptographic hardware by using the underlying ICSF support.

Use one of the following mechanisms to monitor the use of the cryptographic hardware when the IBMJCECCA provider is used:

- ▶ IBM RMF™ - Monitor I CRYPTO report
- ▶ ICSF - DISPLAY ICSF,CARDS
- ▶ ICSF SMF 82 records - subtype 31
- ▶ Hardware event data collection

In this section, we describe how we used the **DISPLAY ICSF,CARDS** command and SMF 82 records to analyze ICSF usage of the cryptographic coprocessors. We did not use RMF because the Monitor I crypto report only analyzes processor usage, which was relatively low in our tests.

Hardware event data collection can also be used to monitor the use of the CPACF facility. It is useful if you want monitor the use of the CPACF by IBMJCE because it does not use the facilities of ICSF.

Running the test

To monitor our cryptographic hardware when TLS was used, we started a Liberty JVM server and ran a workload of 1000 HTTPS requests to our restapp by using the following cipher suite:

```
SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256
```

The Liberty server was configured to disable persistent connections by using the following `httpOptions` element to ensure that each request drove a full TLS handshake:

```
< httpOptions id="httpoptions" keepAliveEnabled="false" />
```

DISPLAY ICSF,CARDS

Note: Use of the ICSF display cards command and SMF82 recording requires updates to the level of ICSF. We used ICSF FMID=HCR77C1.

Before the test, we ran the **/DISPLAY,ICSF,CARDS MVS** command to display the configured cryptographic coprocessors on our system, as shown in Example 6-23.

Example 6-23 DISPLAY ICSF,CARDS: Before test

```
RESPONSE=SC80
CSFM668I 10.30.59 ICSF CARDS 642
ACTIVE DOMAIN = 001
CRYPTO EXPRESS6 COPROCESSOR 6C00
STATUS=Active          SERIAL#=DV785304 LEVEL=6.0.6z
REQUESTS=0000006013 ACTIVE=0000
CRYPTO EXPRESS6 ACCELERATOR 6A01
STATUS=Active
REQUESTS=0000002562 ACTIVE=0000
```

This result confirmed that we have a coprocessor (CEX6C) and an accelerator (CEX6A) configured on the system, and both are active.

After the test, we saw the results of this command that are shown in Example 6-24.

Example 6-24 DISPLAY ICSF,CARDS: After test

```
RESPONSE=SC80
CSFM668I 10.36.25 ICSF CARDS 702
ACTIVE DOMAIN = 001
CRYPTO EXPRESS6 COPROCESSOR 6C00
STATUS=Active          SERIAL#=DV785304 LEVEL=6.0.6z
REQUESTS=0000009040 ACTIVE=0000
CRYPTO EXPRESS6 ACCELERATOR 6A01
STATUS=Active
REQUESTS=0000002639 ACTIVE=0000
```

The difference between these two commands shows that approximately 3000 requests to the CEX6C and 77 requests to the CEX6A during the test period. This result confirms that the cryptographic coprocessors were being used during the test, but we did not fully understand what services were being requested or if the CPACF facility is being driven in our tests.

ICSF SMF 82 records

To gather SMF82 subtype 31 records, we completed the following steps:

1. Before running the test, we started statistics collection in ICSF for engines, services, and algorithms by using the following ICSF command:

```
/SETICSF OPT,STATS=(ENG,SRV,ALG)
```

We then ran the test to drive 1000 request into our restapp.

2. After running the test, we disabled statistics collection to force an SMF record to be collected by using the following command:

```
/SETICSF OPT,STATS=NONE
```

3. We collected SMF82 subtype 31 records from SMF by using the job that is shown in Example 6-25. This job used the IFASMF DL procedure to dump the SMF records from the log stream for the specific day of the year (2017285), sorted them by date, and then formatted out the SMF 82 records by using the new ICSF supplied REXX exec, CSFSMFR.

Example 6-25 ICSF SMF82 subtype: 31 records

```
//CSFSMFJ JOB (999,POK),'FORMAT CRYPTO SMF',CLASS=A,MSGCLASS=T,
// NOTIFY=&SYSUID,TIME=NOLIMIT,REGION=0M
/*JOBPARM L=999,SYSAFF=*
//DUMP EXEC PGM=IFASMF DL
//SYSPRINT DD SYSOUT=*
//DUMPOUT DD DISP=(NEW,PASS),DSN=&&VBS,UNIT=3390,
// SPACE=(CYL,(1,1)),DCB=(LRECL=32760,RECFM=VBS,BLKSIZE=4096)
//SYSIN DD *
LSNAME(IFASMF.DEFAULT,OPTIONS(DUMP))
OUTDD(DUMPOUT,TYPE(82))
DATE(2017285,2017285)
SID(SC80)
SOFTINFLATE
USER1(SMFDPUX1)
USER2(SMFDPUX2)
USER3(SMFDPUX3)
/*
/*-----*
/* COPY VBS TO SHORTER VB AND SORT ON DATE/TIME *
/*-----*
//COPYSORT EXEC PGM=SORT,REGION=200M
//SYSOUT DD SYSOUT=*
//SORTWK01 DD UNIT=VIO,SPACE=(CYL,10)
//SORTIN DD DISP=(OLD,DELETE),DSN=&&VBS
//SORTOUT DD DISP=(NEW,PASS),DSN=&&VB,UNIT=3390,
// SPACE=(CYL,(1,1)),DCB=(LRECL=32752,RECFM=VB)
//SYSIN DD *
SORT FIELDS=(11,4,A,7,4,A),FORMAT=BI,SIZE=E4000
/*
/*-----*
/* FORMAT TYPE 82 RECORDS *
```

```

//*-----*
//FMT      EXEC PGM=IKJEFT01,REGION=200M,DYNAMNBR=100
//SYSPROC DD DISP=SHR,DSN=SYS1.SAMPLIB
//SYSTSPRT DD SYSOUT=*
//INDD     DD DISP=(OLD,DELETE),DSN=&&VB
//OUTDD    DD SYSOUT=*
//SYSTSIN DD *
          %CSFSMFR

```

SMF test data of AES with GCM mode

CSFSMFR produced the report for our test run that is shown in Example 6-26.

Example 6-26 ICSF SMF82 subtype - 31 records.

```

*****
Subtype=001F Crypto Usage Statistics
Written periodically to record crypto usage counts
12 Oct 2017 10:35:01.99
  TME... 003A2397 DTE... 0117285F SID... SC80   SSI... 00000000 STY... 001F
  INTVAL_START.. 10/12/2017 14:30:30.003602
  INTVAL_END.... 10/12/2017 14:35:01.996442
  USERID_AS..... CICSREGN
  USERID_TK.....
  JOBID..... STC08894
  JOBNAME..... SC8CICS7
  JOBNAME2.....
  PLEXNAME..... WTSCPLX8
  DOMAIN..... 1
ENG...CARD...6C00/DV785304... 3003
  ENG...CARD...6A01/N/A    ... 77
ENG...CPACF... 8001
ALG...AES128..... 8000
  ALG...AES256..... 1
  ALG...RSA2048.... 1059
  ALG...RSA4096.... 19
  ALG...ECCP256.... 3000
  ALG...MD5..... 280
  ALG...PRNG..... 10262
  SRV...CSFDSG..... 1000
  SRV...CSFDSV..... 102
  SRV...CSFOWH..... 280
  SRV...CSFPKI..... 1001
  SRV...CSFSYD..... 4000
  SRV...CSFSYE..... 4001
  SRV...CSFIQF..... 3
  SRV...CSFRNGL.... 10262
  SRV...CSFEDH..... 1000
  SRV...CSFPKB..... 3124
*****
Subtype=001F Crypto Usage Statistics
Written periodically to record crypto usage counts
12 Oct 2017 10:35:01.99
  TME... 003A2397 DTE... 0117285F SID... SC80   SSI... 00000000 STY... 001F
  INTVAL_START.. 10/12/2017 14:30:30.003602
  INTVAL_END.... 10/12/2017 14:35:01.996442

```

```

USERID_AS..... CICSREGN
USERID_TK.....
JOBID..... STC08894
JOBNAME..... SC8CICS7
JOBNAME2..... OMVS
PLEXNAME..... WTSCPLX8
DOMAIN..... 1
ALG...PRNG..... 8
SRV...CSFRNG..... 8
*****

```

Note: The SMF output from this test contained two records. Most of the ICSF activity is performed in the CICS address space and contained in the first record; however, the calls to the CSFRNG random number generation service require a space switch from the CICS address space to the ICSF address space and so are contained in another record.

The data in Example 6-26 shows that initialization of the IBMJCECCA security provider and the 1000 TLS handshakes used the following ICSF callable services:

- ▶ CSFDSG - Digital Signature Generate
- ▶ CSFDSV - Digital Signature Verity
- ▶ CSFOWH - One way hash generate
- ▶ CSFPKI - PKA Key Import
- ▶ CSFSYD - Symmetric Key Decipher
- ▶ CSFSYE - Symmetric Key Encipher
- ▶ CSFIQF - ICSF Query Facility
- ▶ CSFRNGL - Random Number Generate
- ▶ CSFEDH - ECC Diffie-Hellman
- ▶ CSFPKB - PKA Token Build

For more information about the ICSF callable services, see the [Resource names for CCA and ICSF entry points topic](#) in IBM Knowledge Center.

The engine statistics (ENG) show 8001 calls to the CPACF to assist with the bulk cipher encrypt/decrypt of the TLS payload. It also showed 3001 calls to the crypto-coprocessor (6C00) and 77 calls to the crypto-accelerator (6A01).

The 8001 calls to the CPACF were driven by 4001 calls to the CSFSYE (symmetric key encipher) and 4000 calls to CSFSYD (symmetric key decipher) services, which resulted in 8000 calls that used the AES128 algorithm. This cipher is the bulk cipher that was specified in our negotiated cipher suite SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256, which uses Elliptic Curve Diffie Hellman with RSA for the key exchange and AES in Galois/Counter Mode (GCM) for the bulk cipher.

Note: In IBM SDK, Java Technology Edition, Version 8, SR5 new functionality was added to the IBMJCE provider to support AES-GCM hardware acceleration on IBM z13® and z14. this support is provided by using the CPACF; therefore, ICSF is not required to obtain the hardware assistance.

For more information, see the [IBM SDK, Java Technology Edition, Version 8, Service Refresh 5 page](#) of the IBM developerWorks website.

SMF test data of AES with CBC mode

Finally, we ran another test workload and modified the cipher suite to use `SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256`. This cipher suite uses the AES cipher-block chaining (CBC) mode of symmetric encryption, which is an older mode of AES that is implemented directly by the IBMJCECCA and IBMJCE providers. It does not require the support of ICSF. The ICSF SMF82 data from this test is shown in Example 6-27.

Example 6-27 ICSF SMF82 subtype: 31 records

```
*****
Subtype=001F Crypto Usage Statistics
Written periodically to record crypto usage counts
12 Oct 2017 11:44:00.69
  TME... 00407445 DTE... 0117285F SID... SC80    SSI... 00000000 STY... 001F
  INTVAL_START.. 10/12/2017 15:42:18.555146
  INTVAL_END.... 10/12/2017 15:44:00.690368
  USERID_AS..... CICSREGN
  USERID_TK.....
  JOBID..... STC08894
  JOBNAME..... SC8CICS7
  JOBNAME2.....
  PLEXNAME..... WTSCPLX8
  DOMAIN..... 1
  ENG...CARD...6C00/DV785304... 3000
  ALG...RSA2048.... 1000
  ALG...ECCP256.... 3000
  ALG...PRNG..... 14000
  SRV...CSFDSG..... 1000
  SRV...CSFPKI..... 1000
  SRV...CSFRNGL.... 14000
  SRV...CSFEDH..... 1000
  SRV...CSFPKB..... 3000
*****
```

In this case, we can see that we now have no use of the CPACF by ICSF because the only engine that is started is the coprocessor (CARD...6C00). The reason for this result is that the AES-CBC mode cipher does not require ICSF support because it is directly implemented by the JCECCA provider that uses native assembler calls to drive CPACF directly.

Instead, we now see that the RSA, ECC, and random number generation algorithms are used in ICSF because they are started during the asymmetric key exchange during the 1000 TLS handshakes in our test case.

This test showed us how the new ICSF SMF statistics provide a powerful tool to analyze the use of the cryptographic hardware by the IBMJCECCA provider when JSSE is used. For more information about the new crypto statistics resource usage, see the [Crypto Statistics Monitor Watches Resource Usage blog post](#) of the IBM Systems Magazine website.



Securing web applications

In this chapter, we describe how to secure web applications that are running in a CICS Liberty JVM server. You learn how to configure Liberty with your preferred security provider, and the various ways you can request credentials from users.

Specifically, we describe the different registry types that are available to Liberty in CICS, along with some of the more popular methods of authentication and authorization.

This chapter includes the following topics:

- ▶ 7.1, “Overview” on page 176
- ▶ 7.2, “z/OS security configuration for Liberty JVM servers” on page 177
- ▶ 7.3, “Configuring a Liberty security registry” on page 180
- ▶ 7.4, “Authentication scenarios” on page 186
- ▶ 7.5, “Authorization scenarios” on page 201
- ▶ 7.6, “Configuring SSO by using Lightweight Third-Party Authentication” on page 214
- ▶ 7.7, “JSON client code with cookie printer” on page 217

7.1 Overview

A high-level overview of security in Liberty JVM servers is shown in Figure 7-1.

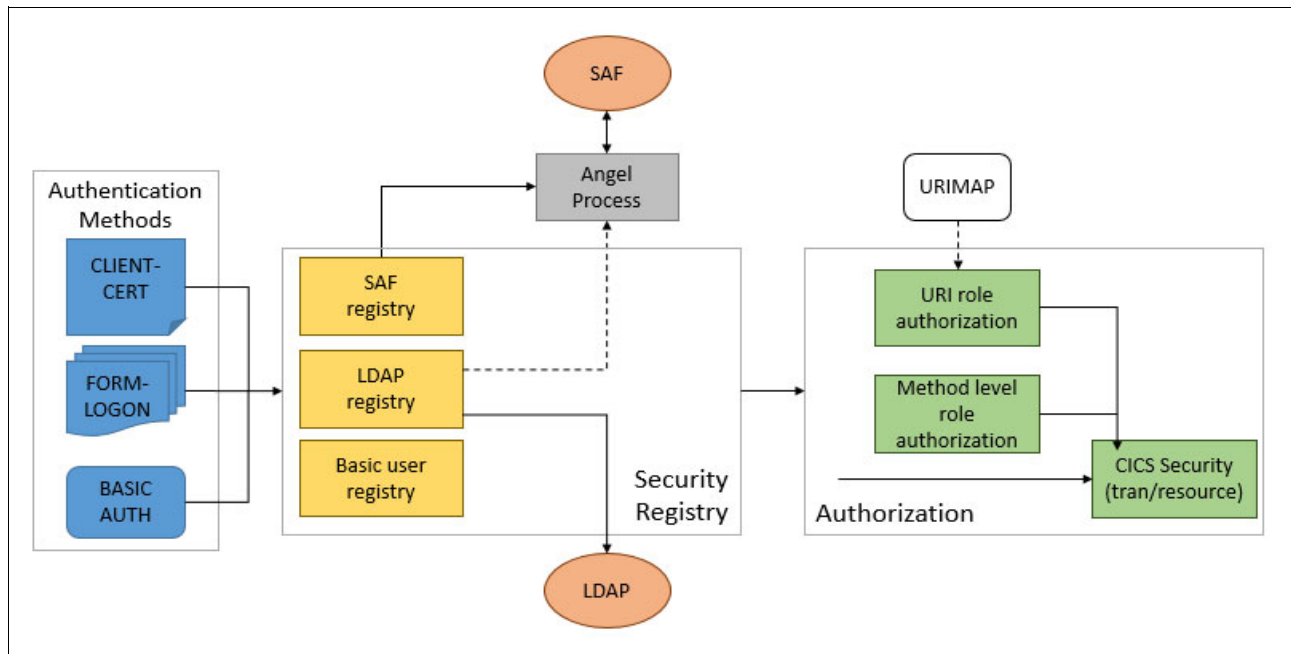


Figure 7-1 Security in Liberty JVM servers overview

As you secure your web applications, consider the following points:

- ▶ Decide whether you are going to use SAF-integrated security. If you are using this security, you must set up the Liberty angel process and some basic security profiles. For more information, see 7.2.1, “Starting the angel process” on page 177.
- ▶ You must decide how you are going to secure the client’s connection with your server. For more information about the use of TLS to achieve connection level security, see Chapter 6, “Configuring Transport Layer Security support” on page 141.
- ▶ Decide which mode of authentication you are going to use in your applications. Each application can use a different mode of authentication. The authentication method is typically outlined in the application’s deployment descriptor (`web.xml`). For more information, see 7.4, “Authentication scenarios” on page 186.
- ▶ Decide on one or more modes of authorization. After users successfully authenticate with the server, several options are available for authorizing them. You use the deployment descriptor (`web.xml`) of your applications to restrict specific URLs. You can use roles that are specified in your source code with EJBROLES in your SAF registry. You can also use CICS transaction and resource security. For more information, see 7.5, “Authorization scenarios” on page 201.
- ▶ Decide whether to use SSO in your Liberty JVM servers. For more information about IBM SSO technology Lightweight Third Party Authentication (LTPA), see 7.6, “Configuring SSO by using Lightweight Third-Party Authentication” on page 214.

7.2 z/OS security configuration for Liberty JVM servers

In this section, we describe the setup of z/OS security for use in Liberty JVM servers. You can skip this section if you are not planning to use SAF security in your CICS Liberty JVM server.

This section, we describe how to set up everything you need and integrate z/OS security with your Liberty JVM server. If you follow the guidelines that are presented in this section, you set up the Liberty angel process and create the security profiles that are required for authenticating users.

7.2.1 Starting the angel process

The Liberty angel process is a started task that allows Liberty to use z/OS authorized services. CICS Liberty JVM servers rely on this process to communicate with a SAF security registry, such as RACF.

The angel process started task JCL includes versions of CICS TS, which support integrated security (that is, CICS TS v5.2 onwards).

You can find this JCL in your CICS USSHOME directory. For our example CICS regions, the following location was used:

```
/usr/lpp/cicsts/cicsts54/wlp/templates/zos/procs/bbgzangl.jcl
```

You must change the JCL before you can run it as a started task. The value of ROOT also must be changed to match your CICS job's USSHOME/wlp. For our example region, the following ROOT value was used:

```
/usr/lpp/cicsts/cicsts54/wlp
```

For each LPAR where you are planning to use a security-integrated CICS Liberty JVM server, you need a started task for the angel process. After the JCL is updated, submit it as a started task.

Note: As of this writing, the angel process cannot communicate at a more granular level than an address space. As a result, only one connection to an angel process from any one address space can be made. Therefore, only one secure Liberty JVM server can be run in a CICS region.

7.2.2 Setting up access to the angel process

Now that the angel task started, security must be configured to allow Liberty JVM servers to connect to the angel process. The examples in this subsection use RACF. If you do not have RACF, run the equivalent commands in your SAF security provider.

The user ID under which the angel process runs requires the SAF STARTED profile to be set up. This process sets the user ID for the started task. In our system, we use the user ID WLPUSER as the user ID for the started task; therefore, we issued the RACF commands that are shown in Example 7-1.

Example 7-1 Creating the required SAF STARTED profile

```
RDEFINE STARTED BBGZANGL.* UACC(NONE) STDATA(USER(WLPUSER))  
SETROPTS RACLIST(STARTED) REFRESH
```

The user ID that is associated with your CICS region's job requires READ access to several profiles in the SERVER class. This access allows Liberty to successfully perform authorization and authentication checks by using the angel process.

To allow the CICS Liberty JVM server to connect to the angel process, you must create and set up access to the SERVER class profile BBG.ANGEL. The commands for our example region whose job runs under the user ID CICSREGN are shown in Example 7-2.

Example 7-2 Setting up and giving access to the server class process BBG.ANGEL

```
RDEFINE SERVER BBG.ANGEL UACC(NONE)
PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(Read) ID(CICSREGN)
```

Next, the CICS JVM server must be allowed to access to the services that are necessary for the CICS Liberty security feature. This access ultimately allows Liberty to use SAF to authenticate and authorize user credentials.

SAF uses several authorized user registries and SAF authorized services to perform authentication and authorization. We must create a RACF profile that is named SAFCREd for these services in the SERVER class. The CICS region's user ID needs READ access to this profile. How this access is set up is shown in Example 7-3 (CICSREGN is our CICS region user ID).

Example 7-3 Setting up the SAF unauthorized services profile

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCREd UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCREd CLASS(SERVER) ACCESS(READ) ID(CICSREGN)
```

In addition to the SAF services, Liberty JVM servers must use the z/OS authorized services. We meet this requirement by setting up the BBGZSAFM profile, which allows the CICS region user ID READ access. The RACF commands that we issued in our system to set up this profile and the required access are shown in Example 7-4.

Example 7-4 Setting up the AUTHMOD.BBGZSAFM profile

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(CICSREGN)
```

Finally, we must create a SERVER profile for the IFAUSAGE services (PRODMGR). We also must give the CICS region user ID READ access. The commands that were issued in our system to perform this step are shown in Example 7-5.

Example 7-5 Creating the profile for the IFAUSAGE services and giving read access

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.PRODMGR UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.PRODMGR CLASS(SERVER) ACCESS(READ) USER(CICSREGN)
```

After creating all of these profiles and setting up access, the SERVER resource must be refreshed by using the command that is shown in Example 7-6.

Example 7-6 RACF refresh command for the SERVER resource

```
SETROPTS RACLIST(SERVER) REFRESH
```

If you want to verify that the angel process was set up correctly, start a CICS Liberty JVM server and open messages.1log. If the angel process was set up correctly, the message that is shown in Example 7-7 on page 179 is displayed in the log.

Example 7-7 Liberty message indicating it successfully connected to the angel process

```
I CWWKB0103I: Authorized service group SAFCREd is available.
```

7.2.3 Profile prefix and required SAF profiles

In this chapter, we describe how to set up and specify access rights for profiles that are specific to our Liberty JVM server. To differentiate between servers and profiles, Liberty uses a profile prefix. The profile prefix is a High Level Qualifier (HLQ) for security profiles, which Liberty uses when looking up profile access rights for user IDs.

Throughout this chapter, we use the profile prefix SC8CICS. The profiles do not need to be unique (several Liberty JVM servers can use the same profile prefix). If you are using only one Liberty JVM server, and therefore one set of profiles, we recommend that you use the APPLID of your CICS region as the profile prefix.

For more information about the steps required to configure a profile prefix, see 7.3.2, “Configuring a SAF registry” on page 182. Some matching profiles in advance must be set up in advance.

First, an APPL class profile must be defined for our prefix and activated. For our SC8CICS prefix, we used the command that is shown in Example 7-8.

Example 7-8 Creating and setting active the SC8CICS profile in the APPL class

```
RDEFINE APPL SC8CICS UACC(NONE)
SETROPTS CLASSACT(APPL)
```

For Liberty to authenticate users, their user IDs need READ access to this profile. This access does not authorize these users to run any programs in Liberty (for more information about setting up this access, see 7.5, “Authorization scenarios” on page 201).

How we granted access to the SC8CICS profile to the user ID WEBUSER is shown in Example 7-9.

Example 7-9 Setting READ access for WEBUSER

```
PERMIT SC8CICS CLASS(APPL) ACCESS(READ) ID(WEBUSER)
```

Liberty uses an unauthenticated user ID to call SAF services to attach the real user IDs. This user ID must be in your system and also needs READ access to the APPL profile. You configure the name of this ID as described in 7.3.2, “Configuring a SAF registry” on page 182.

We use the user ID WSGUEST as our specified unauthenticated user. The WSGUEST ID is the recommended user ID for Liberty servers on z/OS. If you want to use this ID in your system, you must create the user ID.

To grant WSGUEST access to the SC8CICS APPL profile, we run the command that is shown in Example 7-10.

Example 7-10 Granting READ access to WSGUEST

```
PERMIT SC8CICS CLASS(APPL) ACCESS(READ) ID(WSGUEST)
SETROPTS RACLIST(APPL) REFRESH
```

Our CICS region user ID must be granted permission to the WLP z/OS System Security Access Domain (WZSSAD) to make authentication calls for our specified prefix.

To set up this permission, we add a profile for our prefix in BBG.SECPFX and grant READ access to the CICS region user ID. The commands that are shown in Example 7-11 show how we set up the profile for our prefix SC8CICS, and gave our CICS region user ID CICSREGN READ access.

Example 7-11 Setting up the SC8CICS profile for WZSSAD

```
RDEFINE SERVER BBG.SECPFX.SC8CICS UACC(NONE)
PERMIT BBG.SECPFX.SC8CICS CLASS(SERVER) ACCESS(READ) ID(CICSREGN)
SETOPTS RACLIST(SERVER) REFRESH
```

7.2.4 SAF profile summary

Table 7-1 summarizes the RACF profiles we used in this chapter and the permissions we granted to the different user IDs and groups.

Table 7-1 RACF security profiles

Class	Profile	CICS region user ID (CICSREGN)	Unauthenticated user ID (WSGUEST)	Authenticated user IDs or groups (WEBUSER, WEB, ADMN)
Required for angel registration		READ		
SERVER	BBG.ANGEL	READ		
SERVER	BBG.AUTHMOD.BBGZSAFM	READ		
SERVER	BBG.AUTHMOD.BBGZSAFM.SAFCRED	READ		
SERVER	BBG.AUTHMOD.BBGZSAFM.PRODMGR	READ		
Required for authentication or authorization				
SERVER	BBG.SECPFX.SC8CICS	READ		
APPL	SC8CICS		READ	READ
EJBROLE	SC8CICS.com.ibm.cicsdev.restapp.admin SC8CICS.com.ibm.cicsdev.restapp.webusers SC8CICS.com.ibm.cicsdev.restapp.Administrator			READ

7.3 Configuring a Liberty security registry

A security registry is a store of user credentials that is used to authenticate and authorize incoming requests. You can create this registry and specify user IDs and passwords in the server configuration file. It also can be an existing store of user credentials, such as an LDAP active directory or a SAF security registry. The registries are configured through the Liberty JVM server's configuration file (`server.xml`).

In this section, we describe how to set up some simple registries, which can then be used with any of the security models.

The following types of registries are described:

- ▶ Basic user registry: Allows you to specify user IDs and passwords directly in the `server.xml` configuration file.
- ▶ SAF registry: Allows user IDs that are stored in RACF or other SAF security registries to be used for authentication and authorization.
- ▶ LDAP registry: Allows user credentials that are stored in an LDAP server to be used for authentication and authorization.

7.3.1 Configuring a basic user registry

A basic user registry is the simplest registry option that is available. A basic user registry allows you to specify a list of valid users or groups directly in the server configuration file for Liberty.

You can authenticate credentials against this registry in CICS Liberty; however, your applications run under the default CICS user ID or the user ID that is specified in your URIMAP.

Note: If the `cicsts:security` feature is installed, CICS attempts to use your basic user registry for its task attach operation. As a result, if you are planning to use a basic user registry, we recommend that you do not use the `cicsts:security` feature in the same JVM server. Ideally, set `SEC=NO` for the CICS region.

The Liberty angel process is not required for this registry option because authentication is not be tested against your SAF security provider.

Basic user registries list in full the credentials for all of the users and groups that you want to make available to your applications. User names must be specified with passwords, which can be in plain text or encrypted.

Note: Although encoded, passwords are not secure when stored in the `server.xml`. Instead, a user that is viewing the file without knowing the encryption keys cannot read the password. It is recommended that you restrict access to files that contain passwords for any important user ID on the system.

A simple user registry with three users and two groups is shown in Example 7-12.

Example 7-12 Basic user registry in `server.xml` that is used by our server for this scenario

```
<basicRegistry id="basic" realm="customRealm">
  <user name="mitch" password="pAs5w0rd"/>
  <user name="carlos" password="pa$$word!"/>
  <user name="jplaw" password="{xor}Lz4sLCgwLTs="/>
  <group name="webUsers">
    <member name="mitch"/>
    <member name="carlos"/>
  </group>
  <group name="admin">
    <member name="mitch"/>
    <member name="jplaw"/>
  </group>
</basicRegistry>
```

As shown in Example 7-13, users `mitch` and `carlos` use a plain text password. User `jplaw` features an encrypted password that was specified. Users `mitch` and `carlos` are in the group `webUsers`, and the group `admin` contains users `mitch` and `jplaw`. These groups can be used by applications to grant access to certain URLs or to lock out users.

7.3.2 Configuring a SAF registry

SAF is a security provider-neutral interface that is defined by MVS, which allows running programs to use system authorization services to authenticate and authorize user IDs. In our system SAF, a combination with the Liberty angel process allows us to use RACF user IDs to access applications.

SAF is used in Liberty (along with the angel process as described in 7.2, “z/OS security configuration for Liberty JVM servers” on page 177) to authenticate users. These users can be specified in the headers of incoming HTTP requests or within the configuration of your applications.

To configure your Liberty JVM server to use a SAF registry, start by adding the `cicsts:security-1.0` feature to your server configuration’s feature manager, as shown in Example 7-13.

Example 7-13 Enabling the `cicsts:security-1.0` feature in `server.xml`

```
<featureManager>
. . .
  <feature>cicsts:security-1.0</feature>
. . .
</featureManager>
```

This feature enables security integration with CICS, which means that any incoming credentials are authenticated and authorized with your SAF security provider (for example, RACF).

To enable SAF, a set of SAF credentials configuration must be provided in the Liberty JVM server’s configuration file (`server.xml`). As part of this configuration, you can specify something called a *profile prefix*. This prefix is used by the angel process as the HLQ for SAF profiles it must look up. If you do not specify a prefix, the default profile prefix (`BBGZDFLT`) is used.

For our examples, we use the prefix `SC8CICS` because all of our regions APPLIDs start with this prefix. This prefix matches the security prefix SIT parameter (`SECPRFX`) on our CICS region. As a result, our Liberty security profiles names match up clearly with our CICS security profiles. Our SAF credentials configuration resembles the credentials that are shown in Example 7-14.

Example 7-14 SAF credentials for our CICS Liberty JVM server in `server.xml`

```
<safCredentials profilePrefix="SC8CICS"
                unauthenticatedUser="WSGUEST"/>
```

Next, the SAF registry element must be added to your server configuration. The element is simple: You provide is an ID for the XML element. We also provide a value for “realm”, which specifies a name for the registry. You will see the value of the realm attribute appear at times in `messages.log`. The XML element we used is shown in Example 7-15.

Example 7-15 SAF Registry configuration for server.xml

```
<safRegistry id="saf" realm="ITS0"/>
```

The security provider that is defined for our use is not identified. Because SAF is a security provider-neutral interface, this same element works with RACF and any other security provider.

This SAF registry element is all you need to enable your SAF registry. To validate your configuration, you can attempt to connect to the CICS Liberty default application.

The CICS Liberty default application is a simple in-built application, which is automatically installed on start. This application uses the special authorization role CICS_ALL_AUTHENTICATED, which authorizes all authenticated users to call the application. With this role, it also specifies a login configuration of BASIC, which requires callers to include a user ID and password in their HTTP header or by using a browser's prompt.

Note: We recommend disabling the CICS default application for production environments by adding `-Dcom.ibm.cics.jvmserver.wlp.defaultapp=false` to the JVM profile for your Liberty JVM server.

You can access the default application by browsing to the application's address in your browser. In our environment, we use the following URL to access the default application:

```
http://wtsc80:57080/com.ibm.cics.wlp.defaultapp/
```

If you do not know your default application's address, you can find it in the `messages.log` file for your JVM server.

The application requests a user ID and password to log on to the application. Provide a valid user ID that is authorized to log on to CICS. If you followed the set-up process correctly, you see a window similar to the window that is shown in Figure 7-2.

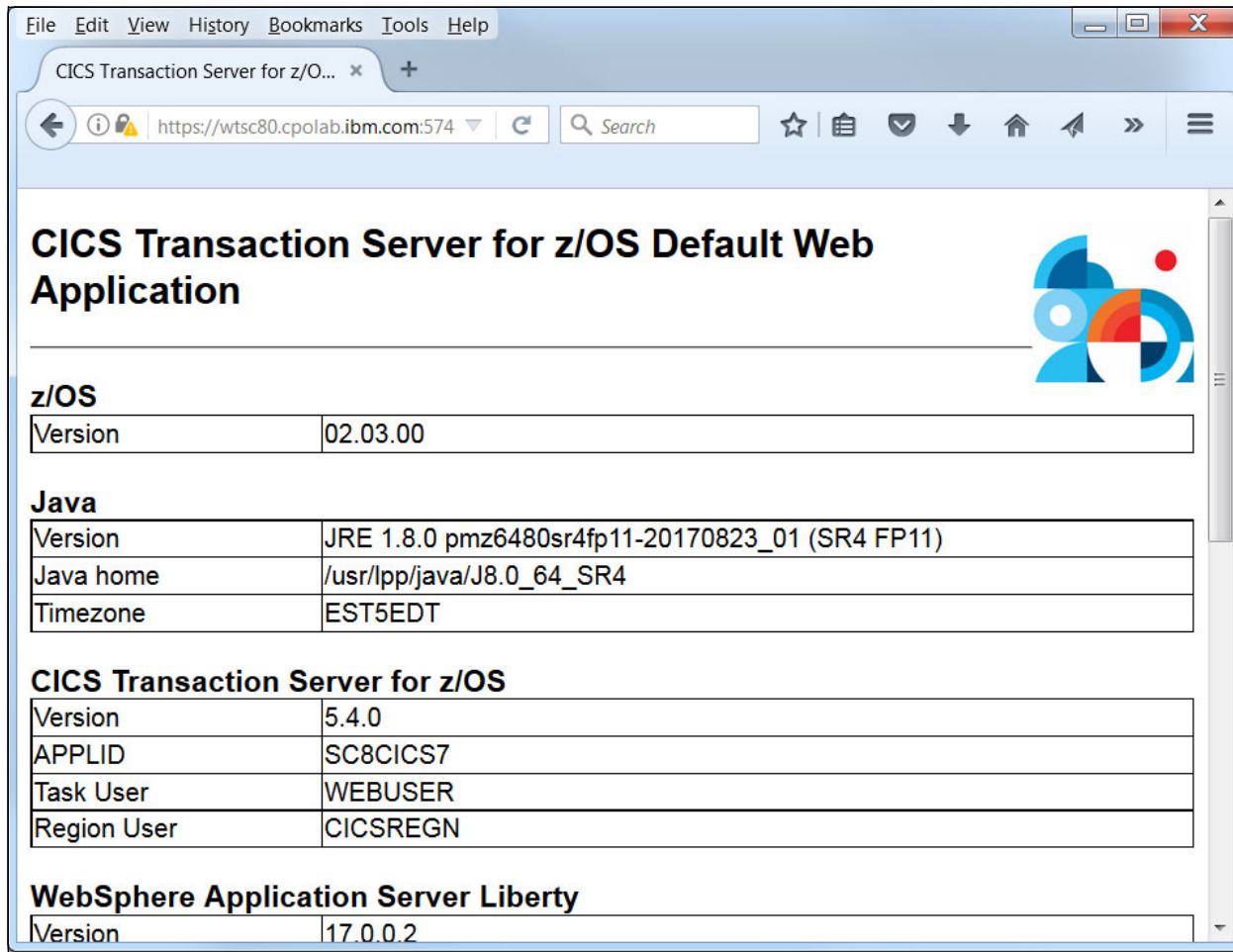


Figure 7-2 Output of the default application Liberty is configured with a SAF registry

7.3.3 Configuring an LDAP registry

Liberty JVM servers in CICS can connect to external LDAP servers and use them to authenticate incoming requests. The connection is made by using HTTP or HTTPS. We recommend the use of HTTPS to connect to the LDAP server where possible to ensure that requests are encrypted.

Note: This scenario requires a basic understanding of LDAP hierarchies because it involves applying filters to directory information trees.

The following options are available for using this registry:

- ▶ Mapping incoming LDAP credentials that are successfully authenticated to a SAF security user ID.
- ▶ Use the LDAP without mapping for authorization, which results in CICS transactions running under the CICS default user or under a user ID that is specified in an applicable URIMAP.

This IBM Redbooks publication focuses on the first option because the second option functions the same way as the basic user registry.

To enable this functionality in CICS Liberty JVM servers, you must add the LDAP registry and distributed identity features to the feature manager's list of features in your server configuration file. The features are shown in Example 7-16.

Example 7-16 LDAP registry feature in Liberty in server.xml

```
<feature>ldapRegistry-3.0</feature>
<feature>cicsts:distributedIdentity-1.0</feature>
```

Note: Liberty continues to develop and improve its features. The version number of this feature can change. Check the latest CICS documentation to determine which versions are supported in your version of CICS.

The `ldapRegistry` feature enables Liberty to connect to a specified LDAP registry over HTTP to authenticate incoming credentials. The `cicsts:distributedIdentity-1.0` enables mapping from LDAP identities to SAF user IDs. After this process is complete, you can add an LDAP registry to your server configuration by using the `ldapRegistry` XML elements, as shown in Example 7-17.

Example 7-17 LDAP registry configuration in server.xml

```
<ldapRegistry baseDn="dc=example,dc=org" bindDn="cn=admin,dc=example,dc=org"
bindPassword="password" host="ldap.hostserver.com" id="ldapRegistry"
ignoreCase="true" ldapType="Custom" port="12345" recursiveSearch="true"
sslEnabled="true" sslRef="Ldap_SSL" realm="LdapRealm">
  <customFilters
    userFilter= "(&(cn=%v)(objectclass=inetOrgPerson))"
    userIdMap="*:cn"/>
</ldapRegistry>

<ssl id="Ldap_SSL" keyStoreRef="LdapKeyStore"
trustStoreRef="LdapTrustStore"/>

<keyStore id="LdapKeyStore" location="{server.config.dir}/LdapKeyStore.jks"
type="JKS" password="{xor}R9MNjj8JL=" />
<keyStore id="LdapTrustStore" location="{server.config.dir}/LdapTrustStore.jks"
type="JKS" password="{xor} R9MNjj8JL=" />
```

Example 7-17 shows the LDAP registry that is for this scenario. The following sets of XML elements are used for the LDAP registry:

- ▶ The `ldapRegistry` element contains all of the information about the target LDAP server, including the base Distinguished Name (DN) and the host address. The `bindDN` and `bindPassword` that are specified in the `ldapRegistry` element. These credentials are used by Liberty to log in to LDAP. The `bindDN` must include access rights to look up other DN's within the specified domain. This `bindDN` often is an admin ID or a user ID that was created specifically for this purpose.
- ▶ The `customFilters` element specifies which filters are applied to incoming user credentials. Specifying this `customFilters` element allows your users to log on with simplified credentials rather than the full LDAP DN.

The elements that are shown in Example 7-17 on page 185 include a filter for user credentials that are coming in to our server. The `userFilter` attribute specifies an object class to be filtered for incoming credentials. In our case, the `inetOrgPerson` object is checked for matching credentials. We also included `&(cn=%v)`, which means that we try to filter our credentials against the Common Name (CN) portion of the DNs that are found in `inetOrgPerson`.

Ultimately, this process results in users needing to provide only the CN portion of their LDAP credentials. If a matching `inetOrgPerson` object exists, the user can log in.

Consider that we use HTTPS with TLS client authentication to connect to LDAP. This configuration is done by specifying `sslEnabled=true` in the `ldapRegistry` element. TLS settings then can be configured by using the `SSL` element. For more information about configuring TLS in CICS Liberty JVM servers, see Chapter 7, “Securing web applications” on page 175.

If you are planning to map LDAP credentials to SAF registry user IDs, you must enable SAF in Liberty. To enable SAF, add the `safCredentials` element to your server configuration file (`server.xml`).

You must specify a profile prefix as part of your SAF credentials. This prefix is used as the HLQ for any SAF profiles that your Liberty JVM server must look up. If you do not specify this prefix, the default prefix is `BBGZDFLT`. For our example, we use `SC8CICS`. The SAF credentials element that we use is shown in Example 7-18.

Example 7-18 SAF credentials configuration in server.xml

```
<safCredentials profilePrefix="SC8CICS"
                unauthenticatedUser="WSGUEST" />
```

After you complete this step, your registry is ready to use. We describe how to use this registry for basic authentication and configure a mapping in 7.4.2, “Basic authentication by using LDAP credentials” on page 190.

7.4 Authentication scenarios

This section describes some of the basic scenarios for configuring authentication in a Liberty JVM server. It also describes some of the different options available to you, and shows you how to configure applications for different authentication mode.

This section does not include instructions for setting up TLS. However, we do recommend that TLS is set up when running these scenarios to ensure that your connection is secure. For more information about setting up TLS, see Chapter 6, “Configuring Transport Layer Security support” on page 141.

By default, Liberty uses Lightweight Third Party Authentication (LTPA) to attempt to optimize the authentication process. For more information about LTPA, see 7.6, “Configuring SSO by using Lightweight Third-Party Authentication” on page 214.

Much of this section involves changing key pieces of configuration in your applications. A login configuration is a piece of XML configuration, which describes the authentication mode that is used by an application. The following settings are described in this chapter:

- **BASIC**, which specifies that callers provide a user name and password. For more information, see 7.4.1, “Basic authentication with a SAF registry” on page 187.

- ▶ FORM, which specifies that an HTML form is provided by the application, which users can use to provide credentials. For more information, see 7.4.3, “Form-based login” on page 193.
- ▶ CLIENT-CERT, which specifies that the TLS/SSL certificate should identify the user. For more information, see 7.4.4, “Certificate-based client authentication” on page 197.

7.4.1 Basic authentication with a SAF registry

This section describes a simple scenario in which we enhance an application for basic authentication by using a SAF registry.

We recommend that TLS is used with basic authentication in Liberty whenever possible. The instructions in this section describe only the setup for basic authentication. For more information about setting up TLS, see Chapter 6, “Configuring Transport Layer Security support” on page 141.

The examples in this section use SAF-based credentials; that is, those credentials that are connected to a SAF security provider. Therefore, we suggest that you complete the process that is described in 7.3.2, “Configuring a SAF registry” before working through this scenario. However, this scenario can be combined with any of the other registries. The security mode is specified at the application level, and is not specific to the registry that is being used by the server.

In this scenario, we use the sample application that is provided by the CICS team on the cicsdev GitHub that is named `cics-java-liberty-restapp-ext`. The source code for this application is available at [the cics-java-liberty-restapp-ext page](#) of the GitHub website.

To enable the basic authentication security mode, we must edit the deployment descriptor for the REST app extensions application. That is, we must add several XML elements to the `web.xml` file in the dynamic web project for the application.

If a project is not yet created for the REST application extensions application, follow the instructions that are described in Chapter 2, “Deploying a web application” on page 27. Ensure that you select the **Generate web.xml deployment descriptor** in the Web Module pane of the New Dynamic Web Project wizard, as shown in Figure 7-3 on page 188.

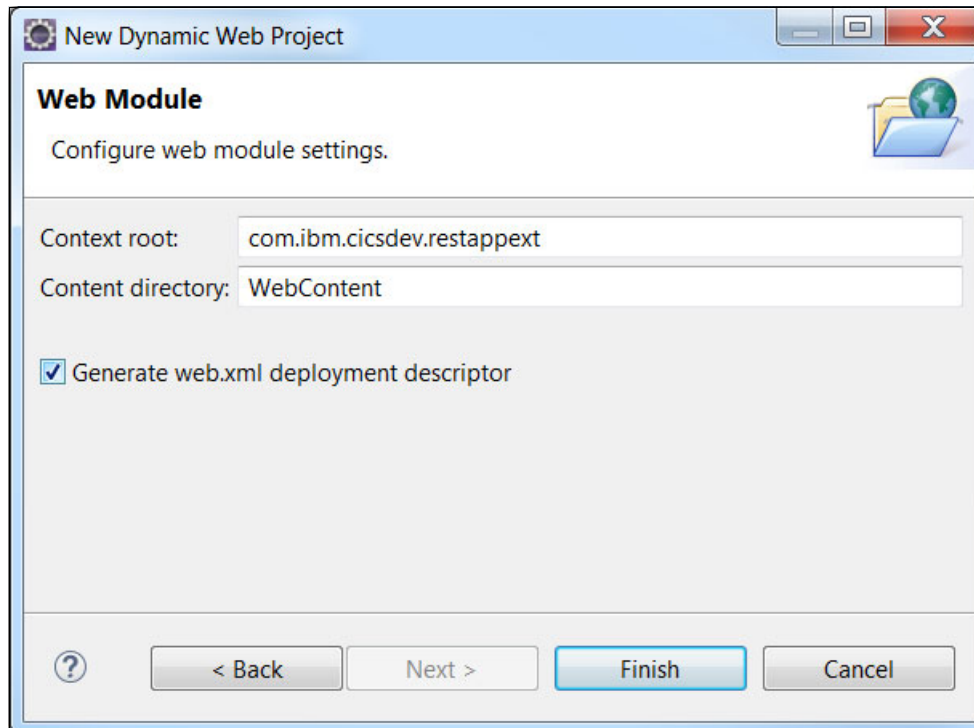


Figure 7-3 Generating `web.xml` by using the New Dynamic Web Project wizard

If you created the project and did not create a `web.xml` deployment descriptor, you can use the Eclipse Java EE tools to generate one. Right-click the dynamic web project and click **Java EE Tools** → **Generate Deployment Descriptor Stub**.

We start by adding some security descriptions into our deployment descriptor file (`web.xml`). First, we must add in a security constraint. The security constraint describes the resources that must be protected in the application and how they are protected. The security constraints configuration that we used in our system is shown in Example 7-19.

Example 7-19 `web.xml` security constraints for enabling basic authentication

```
<security-constraint>
  <display-name>
    com.ibm.cicsdev.restappext_SecurityConstraint
  </display-name>
  <web-resource-collection>
    <web-resource-name>
      com.ibm.cicsdev.restappext
    </web-resource-name>
    <description>
      Protection for urls in restapp
    </description>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>

  <auth-constraint>
    <description>
      All authenticated users of my application
    </description>
    <role-name>cicsAllAuthenticated</role-name>
  </auth-constraint>
</security-constraint>
```

```
</auth-constraint>
```

```
</security-constraint>
```

In Example 7-19 on page 188, consider the url-pattern `/*` in the url-pattern configuration. This pattern specifies that any URL within the application's context-root is subject to the specified security requirements.

The authentication constraint in this case a special role that is named `cicsAllAuthenticated`. This role allows any user who is authenticated to CICS access to the application. CICS creates the role when you install a CICS bundle with access to the role given to anyone who can successfully authenticate with the registry.

Note: The `cicsAllAuthenticated` role is created automatically when you install your application by using CICS bundles. If you intend to install your application by using the application XML element, you must define the role manually.

Finally, you must add a login configuration to the application's `web.xml` file to specify basic authentication as our security mode. The login configuration tells the server which authentication method to use for requests that are coming into the application. The XML elements are shown in Example 7-20.

Example 7-20 Log in configuration in web.xml, which enables basic authentication

```
<login-config>  
  <auth-method>BASIC</auth-method>  
</login-config>
```

Deploy the application to the CICS region by using a CICS bundle, as described in Chapter 2, "Deploying a web application" on page 27. We use the `taskInformation` service to verify that our security configuration is resulting in our user ID being propagated to CICS. The `taskInformation` service returns some basic information to the caller about the CICS TASK that is running for that request, including the associated user ID.

In our environment, we can access this service at the following URL:

```
http://wtsc80:57080/com.ibm.cicsdev.restappext/rest/taskInformation
```

If you are visiting this URL for the first time, you are prompted to log in, as shown in Figure 7-4 on page 190. Enter a valid user ID and password. In our examples, we are use the user ID `WEBUSER` as our login user ID.

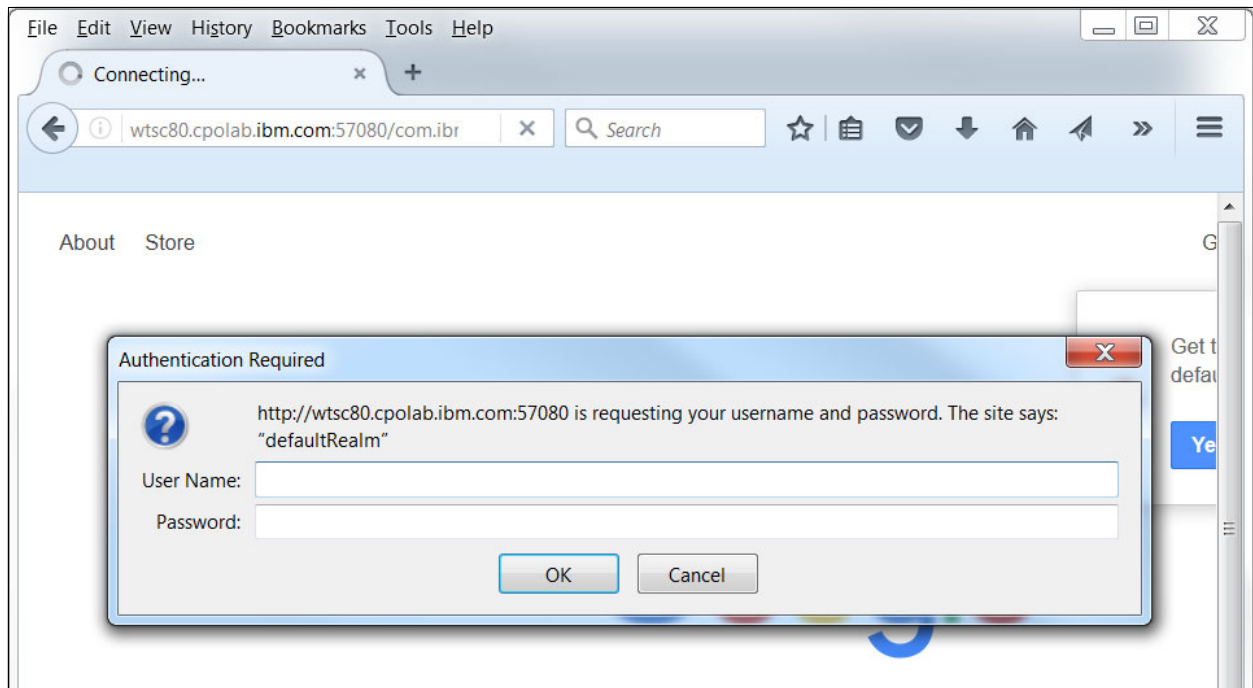


Figure 7-4 Log in prompt from the application

The application returns some basic information about the CICS task that ran the request. If your security was correctly configured, you see the user ID `WEBUSER` as the value of the User ID field. The response from our environment is shown in Example 7-21.

Example 7-21 Example response from the taskInformation service

```
{"transid":"CJSA","userid":"WEBUSER ","tasknum":"16318"}
```

7.4.2 Basic authentication by using LDAP credentials

To complete this scenario, you must set up an LDAP registry in a CICS Liberty JVM server as described in 7.3.3, “Configuring an LDAP registry”.

In this scenario, we use the sample application that is provided by the CICS team on the [cicsdev GitHub](#) that is named `cics-java-liberty-restapp`. The source code for this application is available at [the cics-java-liberty-restapp-ext page](#) of the GitHub website.

To enable the basic authentication security mode, we must edit the deployment descriptor for the restapp application. That is, we must add some elements to the `web.xml` file in the dynamic web project for the restapp application.

If you did not yet create a project for the restapp application, follow the instructions that are described in Chapter 2, “Deploying a web application” on page 27. Ensure that you selected the **Generate web.xml deployment descriptor** option in the Web Module pane of the New Dynamic Web Project wizard, as shown in Figure 7-5 on page 191.

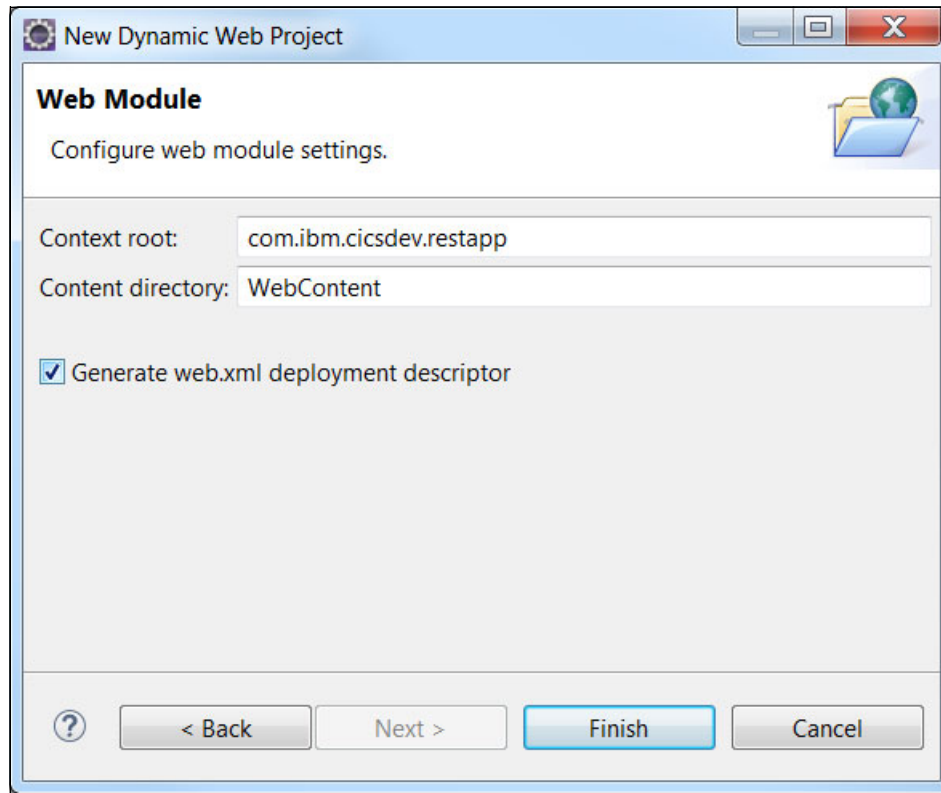


Figure 7-5 Generating web.xml by using the New Dynamic Web Project wizard

If you created the project but did not create a web.xml deployment, you can use the Eclipse Java EE tools to generate one. Right-click the dynamic web project and select **Java EE Tools** → **Generate Deployment Descriptor Stub**.

We start by adding some security descriptions into your deployment descriptor; that is, your web.xml file. We must specify a security constraint in our deployment descriptor, which informs the Liberty server that certain resources are protected. We also must add in a configuration for the login that specifies basic authentication as our login method.

First, we must add in a security constraint. The security constraint describes the resources that must be protected in the application and how they are protected. How the security constraint is set up in our application is shown in Example 7-22.

Example 7-22 Security constraint configuration in web.xml for our application

```
<security-constraint>
  <display-name>
    com.ibm.cicsdev.restapp_SecurityConstraint
  </display-name>

  <web-resource-collection>
    <web-resource-name>
      com.ibm.cicsdev.restapp
    </web-resource-name>
    <description>Protection for urls in restapp</description>
    <url-pattern>*/</url-pattern>
  </web-resource-collection>
```

```

    <auth-constraint>
      <description>
        All authenticated users of my application
      </description>
      <role-name>cicsAllAuthenticated</role-name>
    </auth-constraint>
  </security-constraint>

```

In Example 7-22 on page 191, consider the url-pattern `/*`. This pattern specifies that any URL that is accessed in Liberty that belongs to this application is subject to the specified security requirements.

The authentication constraint in this case is a special role that is named `cicsAllAuthenticated`. This role allows any user who is authenticated to SAF access to the application.

Finally, you must add a login configuration to the application to specify basic authentication as our authentication method. The XML elements that were required to enable basic authentication login are shown in Example 7-23.

Example 7-23 Log in configuration XML elements in web.xml for enabling basic authentication

```

<login-config>
  <auth-method>BASIC</auth-method>
</login-config>

```

Using the value `BASIC` as your `auth-method` value prompts you to log in you attempt to connect to the application by using your browser. We need an LDAP user to test our application. In our environment, we use an LDAP user that is named `LdapUser`. The full DN for the user ID is shown in Example 7-24.

Example 7-24 Our example LDAP user DN

```
cn=LdapUser,dc=example,dc=org
```

In our LDAP server, we use case-insensitive DN's, which is the default behavior for most LDAP servers. However, it can be changed. We recommend that you check whether your DN's are case-sensitive before proceeding.

We need a way of mapping `LdapUser` on to a valid SAF registry ID. We use RACF and RACMAPS because RACF is the security provider on our systems. However, the mapping mechanism works in the same way in other SAF security providers. In our system, we chose to map the `LdapUser` credentials to the SAF registry user ID `LDAPUSER`.

To map `LdapUser` to `LDAPUSER` in RACF, we must set up a RACMAP. This RACMAP lists the full DN for the user ID we plan to map and the LDAP server's host name and the user ID to which we want to map incoming requests. How we set up the mapping for our server `ldaps://ldap.hostserver.com` is shown in Example 7-25.

Example 7-25 RACMAP we created for our LDAP mapping in RACF

```

RACMAP ID(LDAPUSER) MAP
  USERIDFILTER(NAME('cn=LdapUser,dc=example,dc=org'))
  REGISTRY(NAME('ldaps://ldap.hostserver.com'))
  WITHLABEL('Mapping to LDAPUSER')

```

Now, when a user logs in by using the correct credentials for `LdapUser`, they are mapped and the resulting CICS task starts under the user ID `LDAPUSER`.

Although we created a RACMAP with a one-to-one mapping in this scenario, many-to-one mappings can be created.

Our application is now ready to use. Deploy the REST application `com.ibm.cicsdev.restapp`. We can now send an access request to the service along with the LDAP credentials we use to log in. In our system, we use the following URL:

```
http://wtsc80:57080/com.ibm.cicsdev.restapp/rest/cicsinfo
```

If we use a browser, we are prompted for our credentials when we first connect. If we enter our LDAP user name and its corresponding password, we receive an HTTP 200 response and can access the services.

7.4.3 Form-based login

In this section, we describe how to set up the REST application `com.ibm.cicsdev.restapp` that is provided at the `cicsdev` GitHub website. We also describe how to add a restraint to `web.xml`, which enables form-based login.

Form-based login allows you to specify a customized login page to present to users when they attempt to access your applications from a browser.

This scenario uses the sample application that is provided by the CICS team on the `cicsdev` GitHub that is named `cics-java-liberty-restapp`. The source code for this application is available [the cics-java-liberty-restapp page](#) of the GitHub website.

If a project is not yet for the REST application, ensure that you click the **Generate web.xml deployment descriptor** option in the Web Module pane of the New Dynamic Web Project wizard, as shown in Figure 7-6 on page 194.

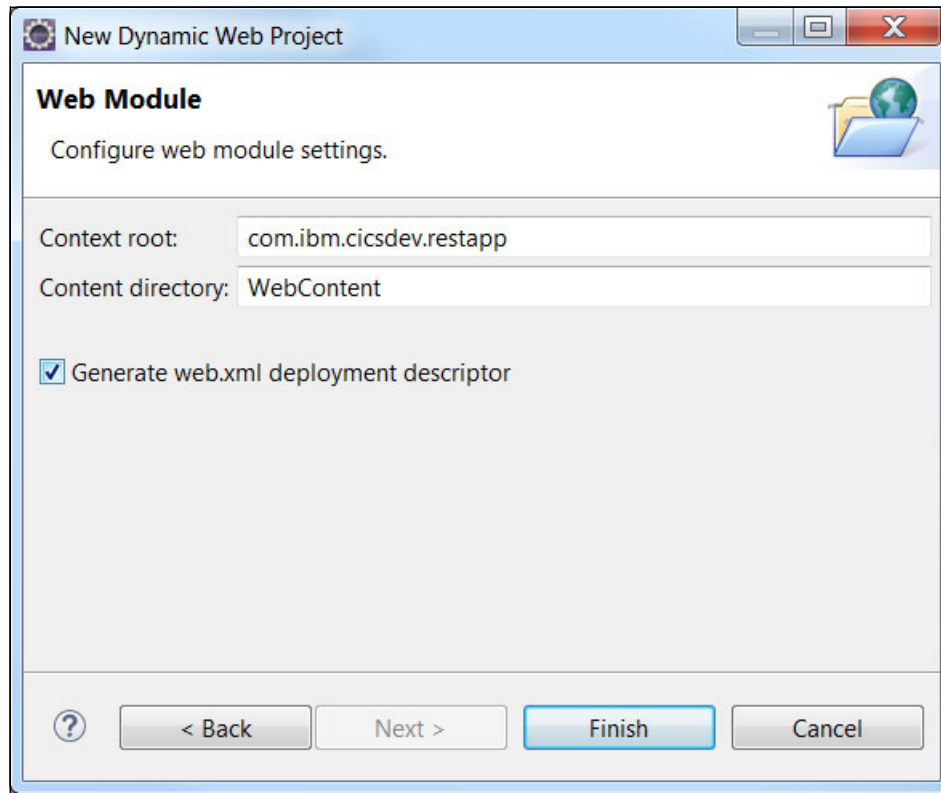


Figure 7-6 Generating web.xml by using the New Dynamic Web Project wizard

We start by creating two HTML pages in our `com.ibm.cicsdev.restapp` project. The first page is our form login page, which is presented to our users when they attempt to access the applications. The second page is an error page that is presented to users who fail to provide valid login credentials.

In our project, we created a file that is named `login.html`, which is stored in the `WebContent` directory of the project. The file `login.html` is a simple page with a form that features input fields for user names and passwords, along with a submit button. The code that we used in our application is shown in Example 7-26.

Example 7-26 The `login.html` page code

```
<html>
  <head><title>Login Form</title></head>
  <body>
    <form action="j_security_check">
      Enter your user name:
      <input type="text" name="j_username"/>
      <br/>
      Enter your password:
      <input type="password" name="j_password"/>
      <input type="submit"/>
    </form>
  </body>
</html>
```

This page displays two simple form entry boxes that allow a user to enter the required information. Because the password field features type password, any information that is entered in that field is masked.

The names and actions that start with `j_` are all keywords for the form login, which tells the application server which fields are to be used for the login.

Next, we create an error page for when the credentials that a user provides are invalid. For our project, we created an HTML page that is named `errorPage.html`. This page returns a text response that notifies the user that the authentication failed. The source code for our error page is shown in Example 7-27.

Example 7-27 errorPage.html file source code

```
<html>
  <head>
    <title>Access Denied.</title>
  </head>
  <body>
    Login Failure. Access Denied.
  </body>
</html>
```

With these two pages created and in place, we can edit our deployment descriptor (`web.xml`) to include form based login. First, we add a security constraint. The security constraint describes the resources that must be protected in the application and how they are protected. The configuration that we used to protect our REST application is shown in Example 7-28.

Example 7-28 Security constraint configuration in web.xml

```
<security-constraint>

  <display-name>
    com.ibm.cicsdev.restapp_SecurityConstraint
  </display-name>

  <web-resource-collection>
    <web-resource-name>
      com.ibm.cicsdev.restapp
    </web-resource-name>
    <description>
      Protection for urls in restapp
    </description>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>

  <auth-constraint>
    <description>
      All authenticated users of my application
    </description>
    <role-name>cicsAllAuthenticated</role-name>
  </auth-constraint>

</security-constraint>
```

As shown in Example 7-28 on page 195, consider the URL pattern `/*`. This pattern specifies that any URL in Liberty that belongs to this application is subject to the specified security requirements.

The authentication constraint in this case is a special role that is named `cicsAllAuthenticated`. This role maps to the `ALL_AUTHENTICATED_USERS` role, which allows any user who is authenticated to CICS access to the application.

Note: The `cicsAllAuthenticated` role is created automatically when you install your application by using CICS bundles. If you intend to install your application by using the application element, you must define the role manually.

Finally, we add a login configuration to the application to specify form-based login as our authentication method and the location of our login and error pages. The configuration that we added to the `web.xml` deployment descriptor is shown in Example 7-29.

Example 7-29 Log in configuration in `web.xml` for form-based login

```
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/login.html</form-login-page>
    <form-error-page>/errorPage.html</form-error-page>
  </form-login-config>
</login-config>
```

The paths to the pages are relative, not absolute. For form-based login, they start from the `WebContent` directory in our dynamic web project.

When we attempt to login to any services that are contained in the `com.ibm.cicsdev.restapp` application, we are presented with the form that we specified in the deployment descriptor, as shown in Figure 7-7.

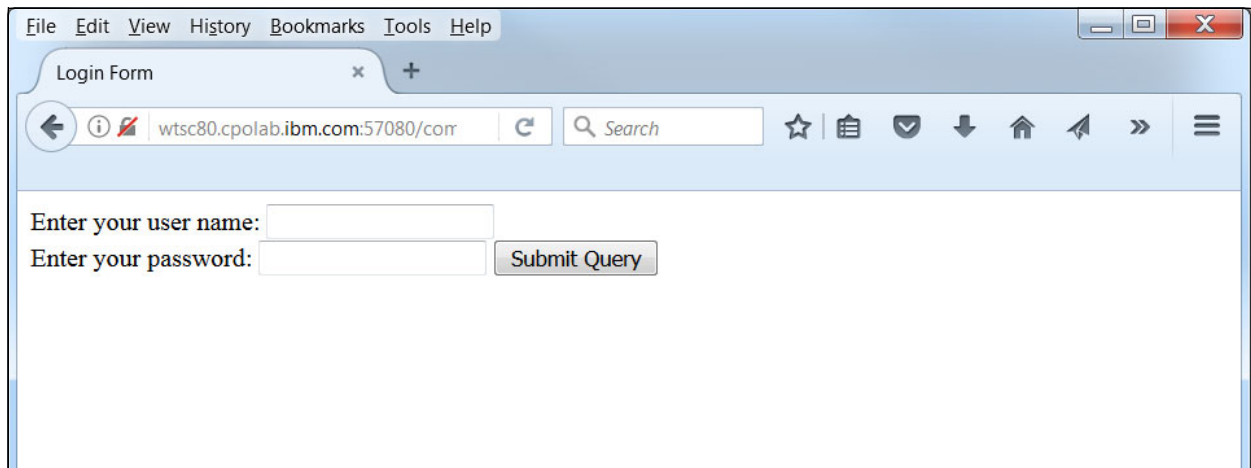
A screenshot of a web browser window. The title bar shows 'File Edit View History Bookmarks Tools Help'. The address bar shows 'Login Form' and the URL 'wtsc80.cpolab.ibm.com:57080/com'. The page content includes two text input fields: 'Enter your user name:' and 'Enter your password:'. To the right of the password field is a button labeled 'Submit Query'.

Figure 7-7 Login form

If we provided incorrect user information (that is, information that normally results in an HTTP 401 response), we see with the error page that we specified, as shown in Figure 7-8.

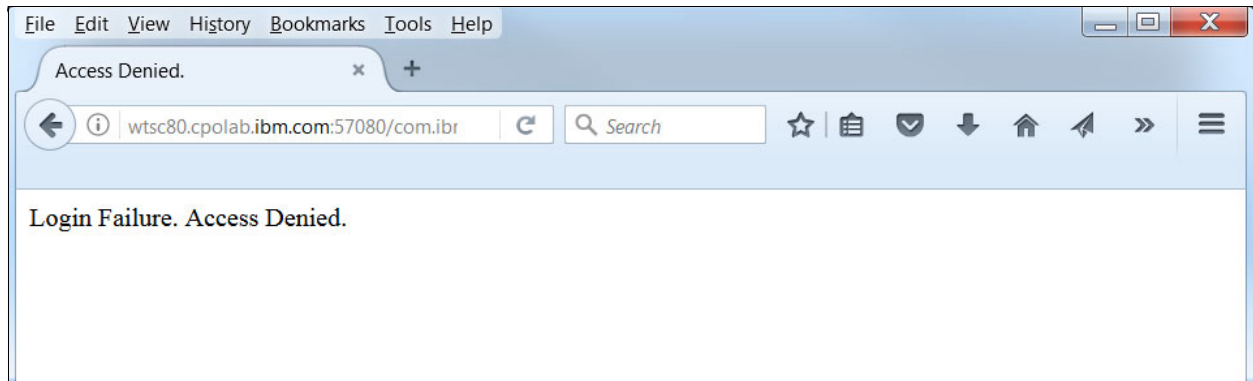


Figure 7-8 Custom error form

7.4.4 Certificate-based client authentication

Certificate-based client authentication allows you to use information that is provided in a client's TLS certificate to identify and map to an associated user ID. It also provides all of the normal benefits that are associated with a secure TLS connection, as described in Chapter 6, "Configuring Transport Layer Security support" on page 141.

As described in Chapter 6, "Configuring Transport Layer Security support" on page 141 we created a certificate for the user ID WEBUSER and stored it in our RACF registry and browser key store. When we use certificate authentication as our authentication method, the certificate that is provided by the client is looked up in the RACF registry. Because we associated the certificate with the user ID WEBUSER in our RACF registry, WEBUSER is the user ID that is used for the CICS task.

To run this scenario, follow the process that is described in 6.4, "TLS client authentication" on page 152. When you complete that section, you should have a Liberty JVM server set up to use two-way TLS handshakes (that is, a connection in which the client and server must provide valid TLS certificates).

You should also have a certificate that is associated with WEBUSER stored in your browser's certificate store on your own workstation. It is a simple step from that point to enable Liberty to use that certificate to resolve the user ID.

For this scenario, we work with the rest app extensions application that is found in the cicsdev GitHub repository, found at the following URL.

<http://wtsc80:57080/com.ibm.cicsdev.restappext/rest/taskInformation>

Download the source code for this application and create a dynamic web project to store it in your Eclipse environment. Ensure that when you are proceeding through the New Dynamic Web Project wizard, you select the **Generate web.xml deployment descriptor** option in the Web Module window, as shown in Figure 7-9 on page 198.

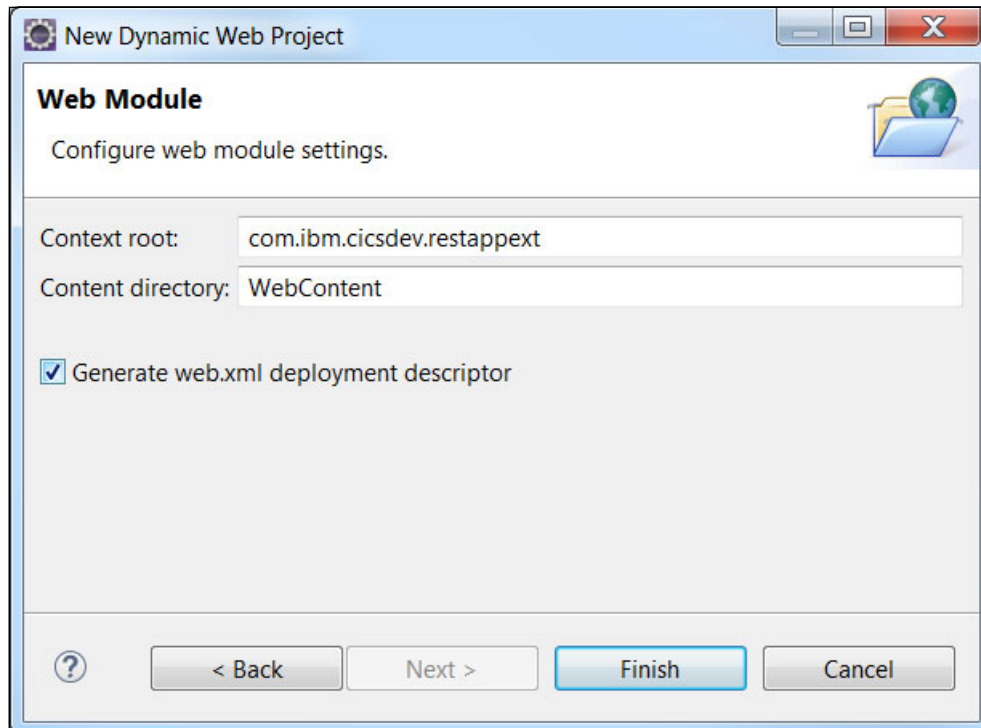


Figure 7-9 Selecting the Generate web.xml deployment descriptor option

Now that we have a project to work with, we must add a security constraint to the deployment descriptor (web.xml). Open the web.xml file and add a security constraint that protects all URLs that are provided by that application. This scenario uses the `cicsAllAuthenticated` role to authenticate users. This role allows anyone who is authenticated and authorized to use CICS to access the application.

The configuration we used to protect all URLs with this role is shown in Example 7-30.

Example 7-30 Protecting all URLs by using the `cicsAllAuthenticated` role in web.xml

```
<security-constraint>

    <display-name>client authentication restraint</display-name>

    <web-resource-collection>
        <web-resource-name>
            authentication.testing.clientauth
        </web-resource-name>
        <description>Protection for urls in the app</description>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>

    <auth-constraint>
        <description>cicsAllAuthenticated role</description>
        <role-name>cicsAllAuthenticated</role-name>
    </auth-constraint>
</security-constraint>
```

Next, we must enable the login configuration for our extended rest application to be CLIENT-CERT. Add the elements that are shown in Example 7-31 to your web.xml deployment descriptor file.

Example 7-31 Log in configuration XML elements in web.xml for enabling client authentication

```
<login-config>
  <auth-method>CLIENT-CERT</auth-method>
</login-config>
```

Now, we must deploy the application to CICS. For this scenario, we install the application by adding the application XML element to the server's configuration file (server.xml).

After you complete the basic configuration for the application is made in your server's configuration file, expand on it to add the *cicsAllAuthenticated* role. Add an application binding configuration for the web application. Then, add the security-role to match. In our server, the configuration that was used is shown in Example 7-32.

Example 7-32 Application binding in server.xml to create the cicsAllAuthenticated role

```
<webApplication id="com.ibm.cicsdev.restappext"
  location="com.ibm.cicsdev.restappext"
  name="com.ibm.cicsdev.restappext">
  <application-bnd>
    <security-role name="cicsAllAuthenticated">
      <special-subject type="ALL_AUTHENTICATED_USERS"/>
    </security-role>
  </application-bnd>
</webApplication>
```

Tip: If you install this application as CICS bundle, the *cicsAllAuthenticated* role is provided for you. Therefore, you do not need to change anything in your CICS Liberty JVM server's configuration file.

The ALL_AUTHENTICATED_USERS subject is a special security subject that specifies access for any user that can be successfully authenticated in the system. However, the user must still have the correct access to run the transaction ID that is associated with Liberty programs that are running in your CICS region. For more information about the use of your SAF registry for securing CICS resources, see 7.5.3, "CICS transaction security with URIMAPs" on page 211.

We can now use the TaskInformation service in our REST application to verify that our user ID is being propagated to the CICS task. Complete the following steps:

1. Ensure that your browser still includes the certificate for WEBUSER that you created when completing the steps that are described in 6.4, "TLS client authentication" on page 152. If the certificate is not included, see 6.4, "TLS client authentication" on page 152 and use the command that shown in that section to export the certificate to your machine by using the **RACDCERT EXPORT** command.
2. Browse to the address of the TaskInformation service in your browser. In our system, we did not change any of the web context root or URI extensions; therefore, we can access our version of the TaskInformation service at the following URL:

<https://wtsc80:57443/com.ibm.cicsdev.restappext/rest/taskInformation>

If this is your first time visiting the URL, you are prompted to select a certificate, as shown in Figure 7-10.

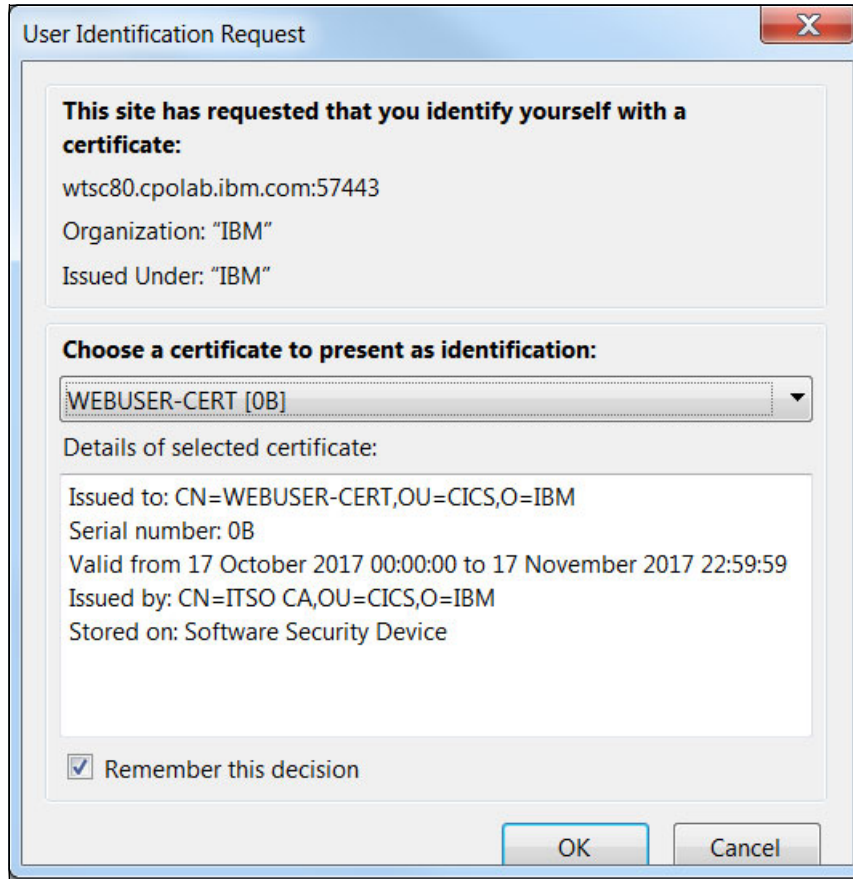


Figure 7-10 Certificate prompt

3. Select the certificate that you created for WEBUSER and select **OK**.

The service returns to your browser some basic information about the CICS task it is running under, including the user ID field. This field shows WEBUSER as our current user ID. The response we received from the service is shown in Example 7-33.

Example 7-33 The response given by the service

```
{"transid":"CJSA","userid":"WEBUSER ","tasknum":"16382"}
```

Enabling fail over to basic authentication

You can authenticate clients who do not have valid certificates by using basic authentication.

To enable this capability, you must add the web container application security element to your Liberty JVM server's configuration file (server.xml). Add the webAppSecurity element to your server.xml file. Then, add the attribute allowFailOverToBasicAuth, with the value set to true. The configuration that we used in our environment to enable this capability is shown in Example 7-34.

Example 7-34 webAppSecurity configuration for failOverToBasicAuth in server.xml

```
<webAppSecurity allowFailOverToBasicAuth="true"/>
```

7.5 Authorization scenarios

In this section, we describe some of the basic scenarios for configuring the authorizing function in CICS Liberty JVM servers.

This section also describes how to use SAF registry groups to restrict access to specific roles by editing an application's deployment descriptor (`web.xml`) and setting up corresponding EJBROLES. For more information, see 7.5.1, "URL-specific authorization by using EJBROLES" on page 201.

We also describe how to set up your SAF registry to authorize roles inside your application code, which were defined by using the `@RolesAllowed` annotation. For more information, see 7.5.2, "Programmatic role authorization by using EJBROLES" on page 206.

Finally, we provide more information about CICS transaction and resource security, which is useful for authorizing users to certain transaction IDs and setting up URIMAPs. For more information, see 7.5.3, "CICS transaction security with URIMAPs" on page 211.

7.5.1 URL-specific authorization by using EJBROLES

In addition to specifying security for entire applications, individual URLs can be secured by using SAF registry groups and EJBROLES. In this scenario, we describe how to use EJBROLES and groups to set up different security levels for different URLs in the same applications.

In this section, we describe how to install an application by using CICS bundles. Although this scenario also works for applications that are installed by using an application element in `server.xml`, it does *not* work for applications that are installed in drop-ins because drop-ins lacks CICS security integration.

To run through this scenario, you must set up a SAF registry. For more information, see 7.3.2, "Configuring a SAF registry" on page 182.

Confirm that the `cicsts:security-1.0` feature is still installed in your `server.xml` file, as shown in Example 7-35.

Example 7-35 Enabling the `cicsts:security-1.0` feature in `server.xml`

```
<featureManager>
  <feature>cicsts:security-1.0</feature>
</featureManager>
```

To configure Liberty use SAF security roles, we must add the `safAuthorization` element to our server's configuration file (`server.xml`). This element is simple; it needs only an ID attribute specified. The configuration we used for this scenario is shown in Example 7-36.

Example 7-36 `safAuthorization` configuration in `server.xml`

```
<safAuthorization id="saf"/>
```

URLs can be restricted in web applications by using specific roles. These roles are outlined by the application in its deployment descriptor (`web.xml`). You can then create corresponding EJBROLES in your SAF registry and grant your user groups access to different sets of groups.

First, you must have an application to which the restrictions are applied. In this scenario, we use the `com.ibm.cicsdev.restapp` application that is [available at the GitHub website](#).

The security constraints are specified as part of the application" deployment descriptor; that is, the `web.xml` file for the corresponding project. If you did not yet create a project for the REST application, ensure that you enable the **generate web.xml deployment descriptor** option.

If you created the REST application project while working through another section of this publication and did not create the deployment descriptor, you can use Eclipse to generate one for you. Right-click the project in Eclipse and then, click **Java EE Tools** → **Generate Deployment Descriptor Stub**.

We now must add our security configuration to the application. First, we add the login configuration to our `web.xml` deployment descriptor file. We add the elements that are shown in Example 7-37 to specify basic authentication.

Example 7-37 Log in configuration in web.xml, which enables basic authentication

```
<login-config>
  <auth-method>BASIC</auth-method>
</login-config>
```

Next, we add two security constraints for our application, which allows us to use different security settings for two different URLs. The first security constraint is protecting the services that are under `InfoResource`; the second protects the service under `ReverseResource`.

By using the configuration that is shown in Example 7-38, the application restricts access to the URL `rest/cicsinfo` so that only users in the role `webusers` have access.

Example 7-38 Security constraint configuration that allows only webusers access

```
<security-constraint>
  <display-name>
    com.ibm.cicsdev.restapp.webusers_Restraint
  </display-name>
  <web-resource-collection>
    <web-resource-name>
      InfoResourceRestraint
    </web-resource-name>
    <description>
      Protection for urls in restapp
    </description>
    <url-pattern>/rest/cicsinfo</url-pattern>
  </web-resource-collection>

  <auth-constraint>
    <description>Web users only</description>
    <role-name>webusers</role-name>
  </auth-constraint>
</security-constraint>
```

Next, we want to restrict the services that are under `/reverse` so that only users in the `admin` role can access it. To enable this restriction, we add the configuration that is shown in Example 7-39 on page 203 to the `web.xml` deployment descriptor for the web application alongside the one for the `webusers`.

Example 7-39 Security constraint configuration in web.xml that limits access to reverseResource

```
<security-constraint>

    <display-name>
        com.ibm.cicsdev.restapp.admin_Reuse
    </display-name>

    <web-resource-collection>
        <web-resource-name>
            ReverseResourceReuse
        </web-resource-name>
        <description>Protection for urls in restapp</description>
        <url-pattern>/rest/reverse</url-pattern>
    </web-resource-collection>

    <auth-constraint>
        <description>Admin only</description>
        <role-name>admin</role-name>
    </auth-constraint>

</security-constraint>
```

Our web application is now configured to handle the roles for our two user groups. We also must create the corresponding roles in RACF.

We create two roles: webusers and admin. The EJBROLES that are used by our applications must be defined in a specific format, as shown in the following example:

```
<profilePrefix>.<application name>.<role name>
```

Consider the following points:

- ▶ profilePrefix must be as specified in the safCredentials element, as shown in Example 7-39. For our example, we use SC8CICS.
- ▶ The application name value must match that of the application as it appears in Liberty. If this value was not changed, it is com.ibm.cicsdev.restapp.
- ▶ The value of role name must match the role that is specified in the deployment descriptor. In our example, we need one EJBROLE each for admin and for webusers.

You can change this format by using the safRoleMapper and profilePattern configuration elements in server.xml.

The following groups are in our system:

- ▶ WEB: Contains all of the user IDs that we expect to use the web applications only.
- ▶ ADMIN: Contains all of the CICS administrator user IDs.

We can use RDEFINE and PERMIT to create and allow access to the EJBROLES for our two groups of user IDs. The format of the command in our system that uses RACF is shown in Example 7-40.

Example 7-40 Setting up the EJB roles for our application

```
RDEFINE EJBROLE SC8CICS.com.ibm.cicsdev.restapp.webusers
      UACC(NONE)
RDEFINE EJBROLE SC8CICS.com.ibm.cicsdev.restapp.admin
      UACC(NONE)
```

```
PERMIT SC8CICS.com.ibm.cicsdev.restapp.webusers CLASS(EJBROLE)
ACCESS(READ) ID(WEB)
PERMIT SC8CICS.com.ibm.cicsdev.restapp.admin CLASS(EJBROLE)
ACCESS(READ) ID(ADMIN)
```

```
SETOPTS RACLIST(EJBROLE) REFRESH
```

Note: In the current version of Liberty, Liberty issues a `RACROUTE REQUEST=LIST` with `GLOBAL=NO` to support earlier versions of z/OS. Therefore, to pick up changes to EJBROLES, you must create the roles before starting CICS or restart CICS after the roles are set up.

If you did not install the REST application as a CICS bundle, follow the instructions that are provided in Chapter 2, “Deploying a web application” on page 27.

Now that our groups and roles are set up, we can demonstrate the functionality by using two user IDs that belong to two different groups. The first user ID `WEBUSER` (see Figure 7-11) can access only the `webusers` role because it is in the `WEB` group. It cannot access the `admin` role because it is not in the `ADMIN` group. The second user ID `MICHAEL` is part of the `ADMIN` group only; therefore, the user ID can access the `admin` role, but cannot access the `webusers` role.

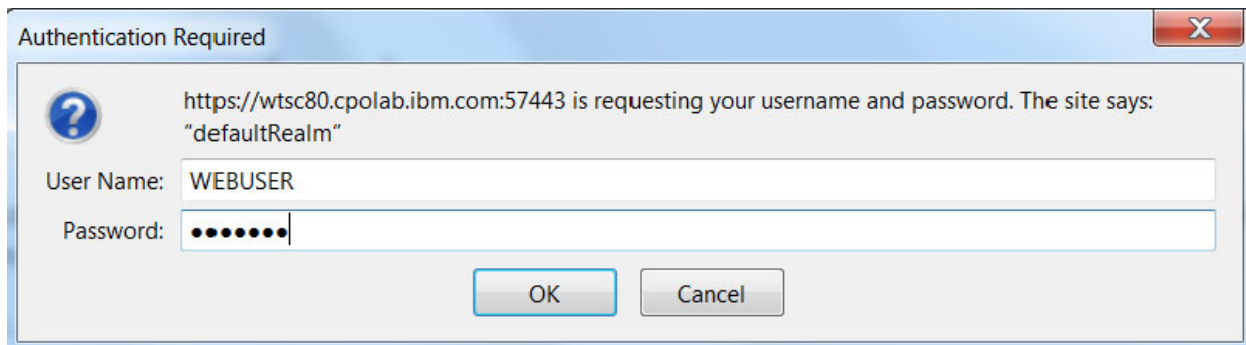


Figure 7-11 Log in prompt for our application by using the user ID `WEBUSER`

No matter which URL we attempt to access, we are prompted for the user ID. The output when the user ID `WEBUSER` is used is shown in Figure 7-12 and Figure 7-13 on page 205. We received an HTTP 200 (request successful) response when we accessed the `cicsinfo` services because the `cicsinfo` service is protected by the `webusers` role. However, accessing the `reverse` services results in an HTTP 403 (unauthorized) response because `WEBUSER` is not in the `admin` group.

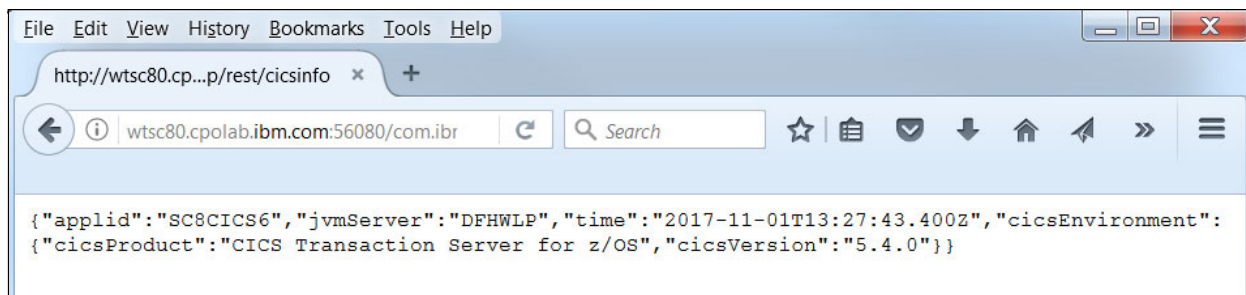


Figure 7-12 Response from `/rest/cicsinfo` for user ID `WEBUSER`

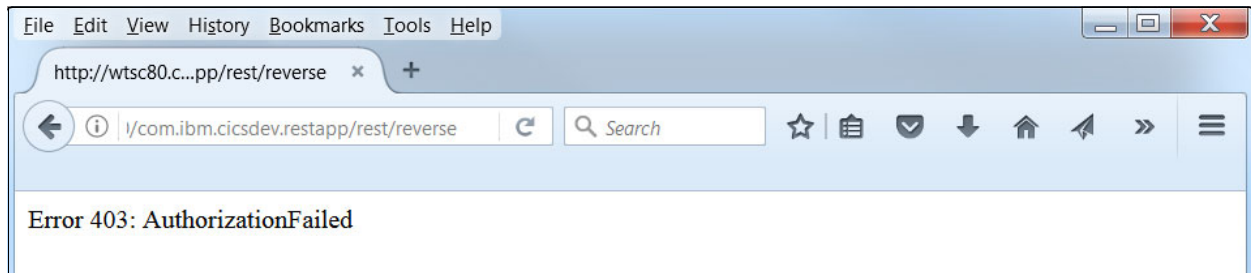


Figure 7-13 Response from /rest/reverse for user ID WEBUSER

In Figure 7-15 and Figure 7-16 on page 206, we can see the behavior when we access the same services by using the user ID MICHAEL (see Figure 7-14). The behavior is reversed to what is shown in Figure 7-12 on page 204 and Figure 7-13. Because MICHAEL is in the admin group and therefore can access the admin role, the user ID can successfully call the reverse service. However, it is not a member of the WEB group; therefore, attempting to access the cicsinfo services returns an HTTP 403 response code.

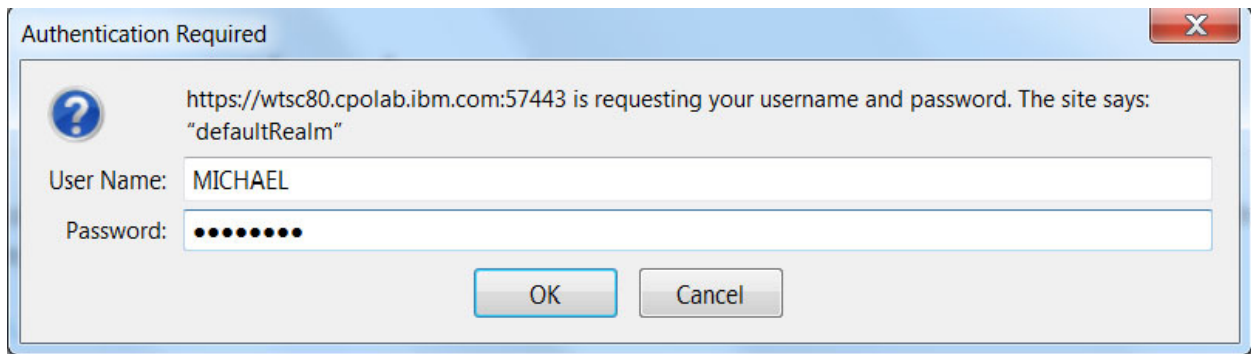


Figure 7-14 Login credentials that use user ID MICHAEL

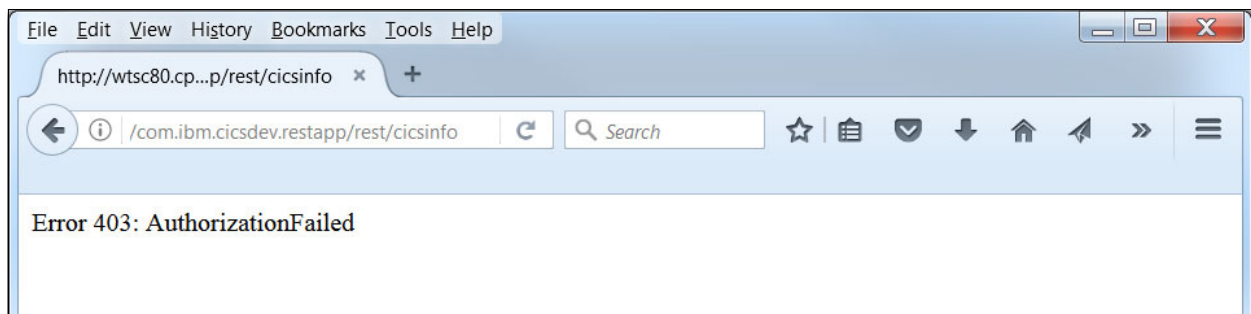


Figure 7-15 Response from /rest/cicsinfo for user ID MICHAEL

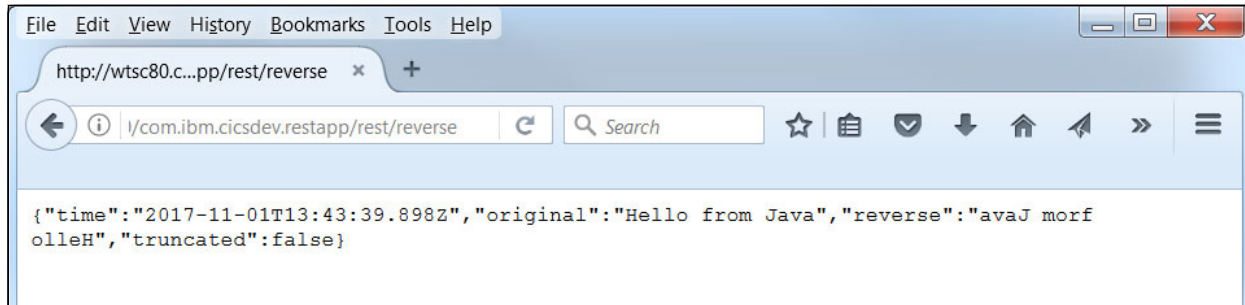


Figure 7-16 Response from /rest/reverse for user ID MICHAEL

7.5.2 Programmatic role authorization by using EJBROLES

Enterprise Java beans (EJBs) give you the option of specifying roles specifically within the EJB code. These roles can be used with Java security roles to restrict access to certain pieces of code at the program level.

You can use programmatic roles with the URL-specific authorization, as described in 7.5.1, “URL-specific authorization by using EJBROLES”. If you choose to use both options, any users need read access to the URL level role and the programmatic role. The URL level role is checked first, followed by the programmatic role when the associated code is first started.

Java security roles are specified users or groups of users that are specified by some part of a Java application, either in the code or within its deployment descriptor. You can use the `@RolesAllowed` annotation to specify specific roles for specific pieces of code before injecting them into other applications (see Example 7-41).

Example 7-41 Example role for Administrator

```
@RolesAllowed("Administrator")
public Item getItem(int id) throws IOException {
    . . .
}
```

EJBROLES are security profiles that are set up in a SAF registry, which can be mapped to Java security roles in your applications.

This scenario guides you through setting up EJBROLES in SAF for use with roles that are specified in your applications. We use an application that is provided by the CICS team that is named `cics-java-liberty-ejb`. This application is available [from the GitHub website](#).

This application is composed of several smaller projects that contain EJBs, which are collected as one Enterprise Archive (EAR). We also included a CICS bundle project, which can be used to export all of the pieces to CICS and install it as a BUNDLE resource.

Before we install the CICS bundle that contains our EJBs, we must update one of the dynamic web projects to include the required security constraints. Several projects are available for the EJB application in your workspace. The project that must be updated is called `com.ibm.cicsdev.ejb.stock.web`.

Open the deployment descriptor for `com.ibm.cicsdev.ejb.stock.webproject`; that is, the `web.xml` file. We add a security constraint that contains our Administrator role and a `login-config` that requires the caller to provide credentials.

Begin by adding a security constraint that affects all URLs in this particular application. The configuration that we used is shown in Example 7-42.

Example 7-42 Security constraint that specifies the Administrator role in web.xml

```
<security-constraint>
  <display-name>
    com.ibm.cicsdev.ejb.stock.web_Restraint
  </display-name>
  <web-resource-collection>
    <web-resource-name>
      com.ibm.cicsdev.ejb.stock.web
    </web-resource-name>
    <description>Protection for all URLs</description>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
</security-constraint>
```

Next, we must add a login configuration that defines the authentication method to be used. For the purposes of this scenario, we use basic authentication (that is, authentication that features a user name and password). To specify basic authentication, we can add the XML elements that are shown in Example 7-43 to the deployment descriptor with the security constraint we just added.

Example 7-43 Log in configuration XML elements in web.xml for enabling basic authentication

```
<login-config>
  <auth-method>BASIC</auth-method>
</login-config>
```

We also must update our Liberty JVM server's server configuration by adding features to support roles, SAF security, JAX-RS, and EJB-Lite. The following features must be added into the feature manager list in the `server.xml` file:

- ▶ `<feature>ejbLite-3.2</feature>`
- ▶ `<feature>jaxrs-2.0</feature>`
- ▶ `<feature>cicsts:security-1.0</feature>`

Next, we must configure Liberty to use a SAF registry to authenticate incoming requests. We add the following XML elements:

- ▶ `safRegistry`: Specifies that we use SAF as our security registry
- ▶ `safCredentials`: Specifies the basic SAF information
- ▶ `safAuthorization`: Specifies that we use SAF roles and not Java security roles

All of these elements are simple pieces of configuration. How we configured our SAF settings is shown in Example 7-44.

Example 7-44 SAF configuration in server.xml for this subsection

```
<safRegistry enableFailover="false" id="saf" realm="ITS0"/>
<safCredentials profilePrefix="SC8CICS"
  unauthenticatedUser="WSGUEST"/>
<safAuthorization id="saf"/>
```

The `profilePrefix` attribute in the `safCredentials` element specifies the HLQ for the SAF profile. When we define our `EJBROLE`, we must ensure that this element is the HLQ with the rest of the role following shortly afterward.

Note: If you do not specify a profile prefix in your server configuration, a default prefix of BBGZANGL is used instead when matching a role with EJBROLEs. Accordingly, you must adjust your SAF profiles.

The EJBROLEs that are used by our applications must be defined in a specific format, as shown in the following example:

```
<profilePrefix>.<application name>.<role name>
```

Consider the following points:

- ▶ `profilePrefix` must be as specified in the `safCredentials` element, as shown in Example 7-44 on page 207. For our example, we use SC8CICS.
- ▶ The `application name` value must match the application as it appears in Liberty. If you did not change any of our projects, by default this name is `com.ibm.cicsdev.ejb` because the EJB that contains the restriction is contained within that project.
- ▶ The value of `role name` must match the role that is specified in the EJB. In our example, you can see the value of this role that is annotated on the class `com.ibm.cicsdev.ejb.CatalogueBean`.

You can change this format by using the `safRoleMapper` and `profilePattern` configuration elements in `server.xml`.

Use **RDEFINE** and **PERMIT RACF** commands to create and allow access to the EJBROLE for user ID WEBUSER. The format of the command in our system that uses RACF is shown in Example 7-45.

Example 7-45 Setting up the EJB role for our application

```
RDEFINE EJBROLE SC8CICS.com.ibm.cicsdev.ejb.Administrator
      UACC(NONE)

PERMIT SC8CICS.com.ibm.cicsdev.ejb.Administrator CLASS(EJBROLE)
      ACCESS(READ) ID(WEBUSER)
```

Note: In the current version of Liberty, Liberty issues a `RACROUTE REQUEST=LIST` with `GLOBAL=NO` to support earlier versions of z/OS. To pick up changes to EJBROLEs, you must create the roles before starting CICS or restart CICS after the roles are set up.

Now that everything is set up, we can start our application. In this scenario, we assume that the application is installed as a CICS bundle. The projects you downloaded from the `cicsdev` GitHub website include a CICS bundle project that is `com.ibm.cicsdev.ejb.bundle`. For more information about installing this application as a CICS bundle, see Chapter 2, “Deploying a web application” on page 27.

After the application is started, we can make a call to the EJB store front to see what items are listed. The store front on our systems uses the following URL:

```
http://wtsc80:57080/shop/
```

On first call, you find the shop is empty. We must add stock to the catalog, which is an operation that is protected with the @RolesAllowed(Administrator) role in the corresponding EJB (see Figure 7-17).

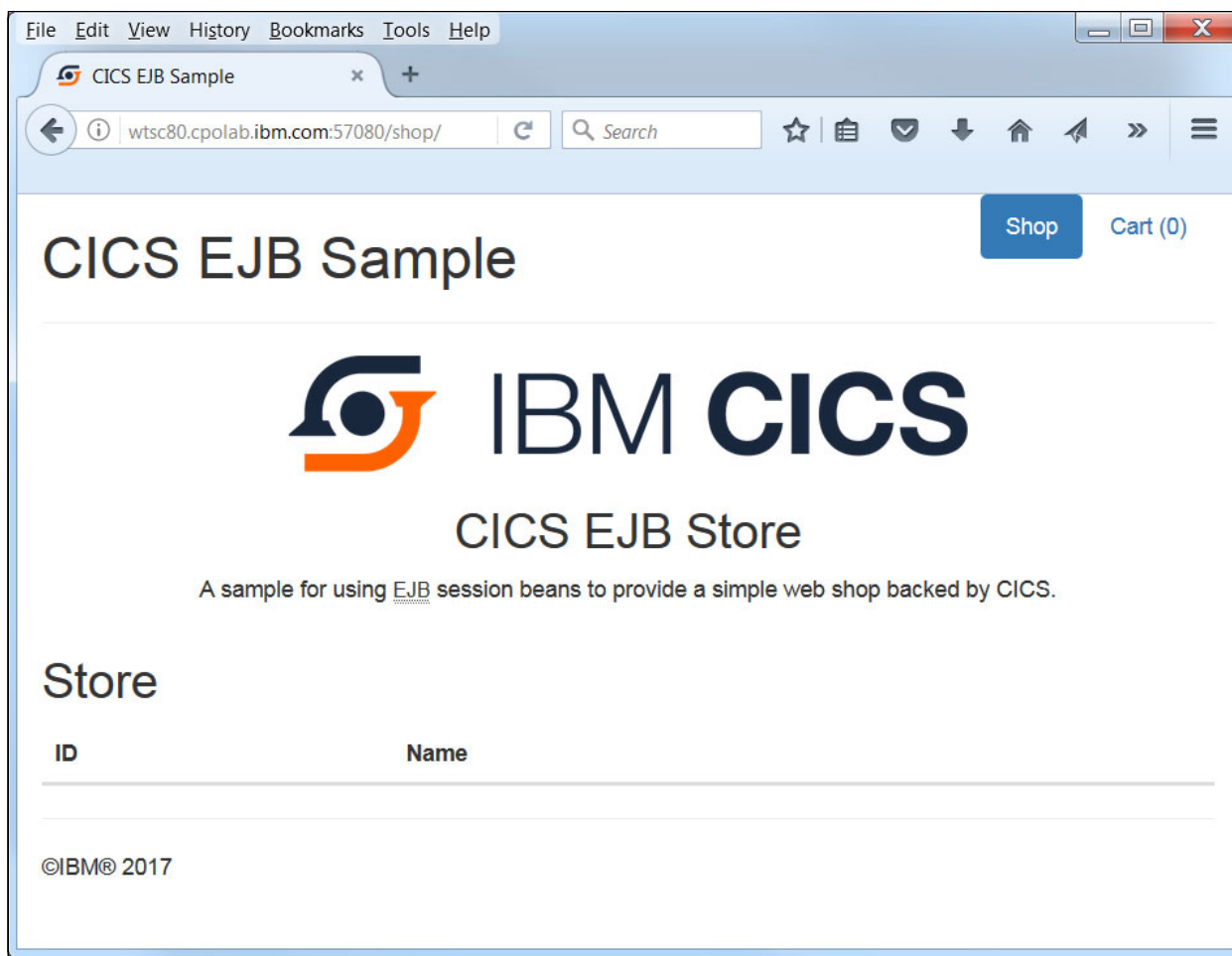


Figure 7-17 Empty EJB sample shop

To add stock, we must make a call to the `createItem` REST service that is provided by the EJB shop example. This particular service uses methods that are protected by the Administrator role; therefore, we must send credentials for WEBUSER by using HTTP POST with our JSON data.

We are sending the JSON data that is shown in Example 7-46. This data causes the web shop to update by adding an item that is named `Golf Club` to its stock list.

Example 7-46 JSON data used by our HTTP POST request.

```
{"name": "Golf Club", "stock": 1}
```

The following methods can be used to send data to the URL by using HTTP POST:

- ▶ You can use a JAX-RS browser extension to handle the request. Multiple options are available for most browsers. Ensure that the extension you use can handle authentication headers; otherwise, you receive a 401 response code from the server.
- ▶ You can use `curl` if it is installed on your system. Specify the JSON data with the user credentials and URL.

- You can call the service from Java. We provided a sample program that can send an HTTP POST request to a specified URL with some JSON data. This sample requires only base Java 8 to run (no other libraries are required). You must update the code to use your host name, ports, and servlet path (if you changed it) and include your own user credentials.

Now that the stock list is updated for the shop, we can reissue a request to the store front to view the items in the shop. If your previous JAX-RS request was successful, you should see the Golf Club item listed, as shown in Figure 7-18.

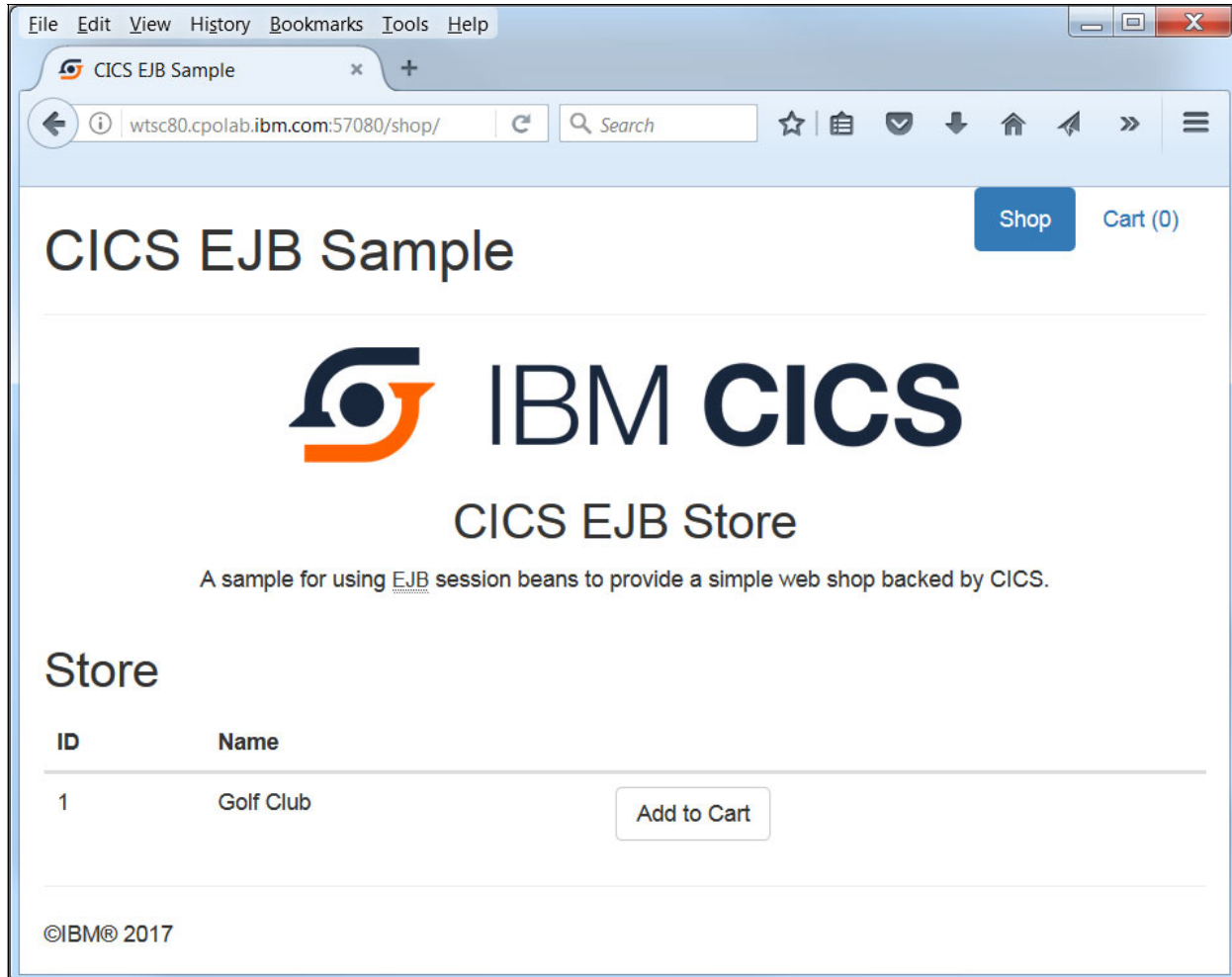


Figure 7-18 Shop front after added stock is added

7.5.3 CICS transaction security with URIMAPs

In addition to the web security that is provided by Liberty, applications that are running in Liberty JVM servers are subject to CICS transaction and resource security. Any user IDs that are authenticated in Liberty must also include the correct access rights to run transactions, access files, or call programs.

This section describes CICS security for web applications in a Liberty JVM server.

By default, HTTP requests that are received by Liberty run under the CICS Liberty default transaction ID of CJSA. This scenario guides you through the steps you must complete to change the transaction ID and set up CICS transaction security for your new transaction ID.

We use the REST extensions application that is provided by the CICS team as our example application in this chapter. The project is available from [the cics-java-liberty-restapp-ext page](#) of the GitHub website.

Install the project as a CICS bundle in your region. For more information, see Chapter 2, “Deploying a web application” on page 27.

We must define a new transaction for our URIMAP to use. Although we use the transaction ID JURI, you can use any code that is not in use. To define and install our JURI transaction, we used the command that is shown in Example 7-47.

Example 7-47 CEDA commands defining and installing our JURI transaction in CICS

```
CEDA DEFINE TRANSACTION(JURI) GROUP(ITSOWLP)
      PROGRAM(DFHSJTHP)
```

```
CEDA INSTALL TRANSACTION(JURI) GROUP(ITSOWLP)
```

You receive a warning when defining this transaction because the value of the PROGRAM attribute started with the letters DFH. This warning can be safely ignored. The program DFHSJTHP acts as a dummy program in this scenario. It does not run any code. Instead, it allows your Java code to run under the specified transaction.

Now that your transaction and application are enabled in CICS, we can define a new URIMAP for any requests that are coming in to the application. Log on to CICS and use CEDA to define a new URIMAP.

The URIMAP must apply to the path `*/TaskInformation` on the HTTP and the HTTPS ports for your CICS Liberty JVM server. The command structure for our JVM server and application is shown in Example 7-48.

Example 7-48 CEDA commands for defining and installing our URIMAP for CICS Liberty

```
CEDA DEFINE URIMAP(RESTMAP) GROUP(ITSOWLP)
      PATH(*/TaskInformation)
      SCHEME(HTTP) USAGE(JVMSEVER)
      HOST(wtsc80)
      PORT(*) TRANSACTION(JURI)
```

```
CEDA INSTALL URIMAP(RESTMAP) GROUP(ITSOWLP)
```

Note: URIMAPs that specify SCHEME(HTTP) are applied to HTTP and HTTPS requests. If you specify SCHEME(HTTPS), the URIMAP is applied to HTTPS requests only.

After they are installed, any web requests coming in to this application in Liberty run under the transaction ID JURI instead of CJSJ. The next step is to create a RACF profile for our new transaction and give WEBUSER read access.

To create this profile and grant access, create a TCICSTRN profile for JURI and run a **PERMIT** command for WEBUSER. The commands that we used are shown in Example 7-49.

Example 7-49 Setting up security for JURI

```
RDEFINE TCICSTRN SC8CICS.JURI NOTIFY(CICSREGN) UACC(NONE)
PERMIT SC8CICS.JURI CLASS(TCICSTRN) ID(WEBUSER) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

Our JURI profile is prefixed with SC8CICS. This security prefix is used for our production regions. You must include any security prefix that you use. The security prefix for your CICS region is set by using the SECPRFX SIT parameter.

The service TaskInformation in the application `com.ibm.cicsdev.restappext` returns to the caller some basic information about the TASK that is assigned to it in CICS. We use this application to establish that our URIMAP was successfully applied and check whether WEBUSER can run the JURI transaction. To perform this check, we use basic authentication, as described in 7.4.1, “Basic authentication with a SAF registry” on page 187.

To verify that your URIMAP is configured correctly, make a call to the TaskInformation service from your browser. In our setup, the following URL is used:

`http://wtsc80:57080/com.ibm.cicsdev.restappext/rest/TaskInformation`

You should receive JSON data that resembles the data in Example 7-50.

Example 7-50 Example response from the TaskInformation service

```
{"transid":"JURI","userid":"WEBUSER ","tasknum":"16212"}
```

Our URIMAP was applied successfully because the `transid` field returned the value of JURI and not CJSJ.

Setting a user ID for unprotected applications

Not every application in Liberty might be running with security constraints that are specified in their deployment descriptor. This scenario describes how to assign a user ID to requests that are coming into an application without security constraints, by using a URIMAP in CICS. For this scenario, we use the REST extensions application that is available from [the cics-java-liberty-restapp-ext page](#) of the GitHub website.

Because you do not need to change this application, it can be deployed as described in Chapter 2, “Deploying a web application” on page 27. Ensure no URIMAPs are active in your CICS region; for example, the URIMAP RESTMAP created earlier in 7.5.3, “CICS transaction security with URIMAPs” on page 211.

For the purposes of this section, it is assumed that your CICS region is running with security active; that is, the SIT parameter SEC=YES is set for your CICS region.

Running the unsecured REST extensions application under a specific user ID is easy to accomplish. Only a matching URIMAP must be installed. We use the TaskInformation service from the REST extensions application. Therefore, we must make its URL the basis of the URL that is specified in the URIMAP.

We also must specify the host machine for the CICS region. In our CICS region, we defined and installed our URIMAPs by using the commands that are shown in Example 7-51.

Example 7-51 CEDA commands for defining and installing our URIMAP for CICS Liberty

```
CEDA DEFINE URIMAP(RESTMAP) GROUP(ITSOWLPL)
          PATH(* /TaskInformation)
          SCHEME(HTTP) USAGE(JVMSEVER)
          HOST(wtsc80)
          PORT(*) USER(UMAPUSER)
```

```
CEDA INSTALL URIMAP(RESTMAP) GROUP(ITSOWLPL)
```

The USER attribute for the URIMAP is set to UMAPUSER. This attribute redefines the user ID to be UMAPUSER when the CICS task is attached. In this scenario, we do not change the transaction ID; therefore, the user ID UMAPUSER must be granted access to CJSJ to run.

Now, we can check the outcome of our URIMAP by calling the TaskInformation service. In our system, we access this service at the following URL by using an HTTP GET request:

`http://wtsc80:57080/com.ibm.cicsdev.restappext/rest/taskInformation`

The service returns information about the user ID and transaction ID that is used by the request. You receive get a response that includes the user field specified as UMAPUSER. The response from the service that is running in our system is as shown in Example 7-52.

Example 7-52 Response from the taskInformation service with the URIMAP in place

```
{"transid":"CJSJ","userid":"UMAPUSER","tasknum":"16587"}
```

7.6 Configuring SSO by using Lightweight Third-Party Authentication

Lightweight Third-Party Authentication (LTPA) is an IBM single-sign on technology that reduces the number of times a user's credentials are checked against a security registry. When a new authentication request occurs, the user ID and password (or other credentials) are authenticated as normal, but on response the server returns a signed authentication token to the requester, as shown in Figure 7-19.

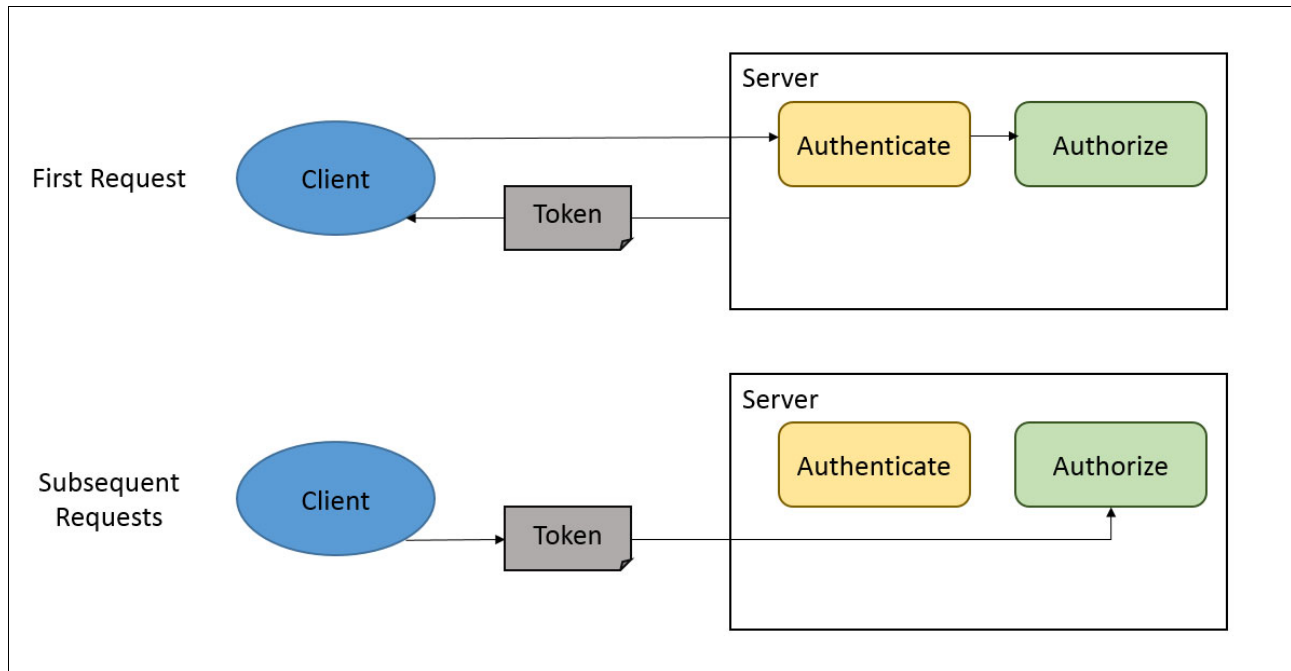


Figure 7-19 Basic authentication flow by using identity tokens

The client can then provide this token in subsequent requests. If the token is still valid, the server uses it to derive the user ID, skip the authentication step, and proceed to authorization.

This configuration can simplify the authentication process in systems where components that must communicate with one another are spread across multiple environments. Particularly, where users are required to enter credentials manually.

If you use the `cicts:security-1.0` or `appSecurity-2.0` features in your Liberty JVM server, Liberty uses LTPA tokens by default in its authentication process. Next, we describe how to configure these tokens, how to disable them, and how to ensure that they are sent over SSL only.

7.6.1 Configuring LTPA

LTPA is easy to set up. First, you must enable an authentication mode for use in your applications. The authentication mode is not too important because LTPA is compatible with most modes.

For the purposes of using tokens for the first time, we recommend the use of basic authentication because it is the easiest to work with and does not require setting up TLS certificates. After the basic scenario is working, you can add TLS to secure the connection between the client and server.

LTPA does not require any other features to be added to the server configuration file (`server.xml`). An LTPA configuration XML element is added to your configuration. The LTPA keys configuration we used is shown in Example 7-53.

Example 7-53 Example configuration in `server.xml` for LTPA. Password is encoded.

```
<ltpa keysPassword="{xor}Lz4sLCgwLTs="
keysFileName="${server.output.dir}/resources/security/ltpa.keys"
/>
```

We chose to keep the keys in the `resources/security` directory. You do not need to create these keys in advance. Liberty automatically creates the keys for you if the keys do not exist.

If you do not change the `keysFileName` attribute, the keys are stored in your server's output directory/`resources/security/ltpa.keys`.

When you successfully connect and authenticate to a Liberty application, the server automatically generates and returns an LTPA token as a cookie to the requester. When the requester sends another HTTP request to the same server, it can return the LTPA token as a cookie. The server validates and decrypts the token and extracts the attached user ID by using it for the request.

The cookie that is stored is named `LtpaToken2`. The following methods can be used to review this cookie:

- You can use your browser to connect to the application. After you successfully authenticate and call the backing service, you can use your browser to view the cookies that are stored locally on your workstation.

The option to view cookies often can be found in your browser's settings menus. In Firefox, click **Settings** → **Privacy** → **remove individual cookies**. Then, enter `LtpaToken2` into the Search field, as shown in Figure 7-20 on page 216.

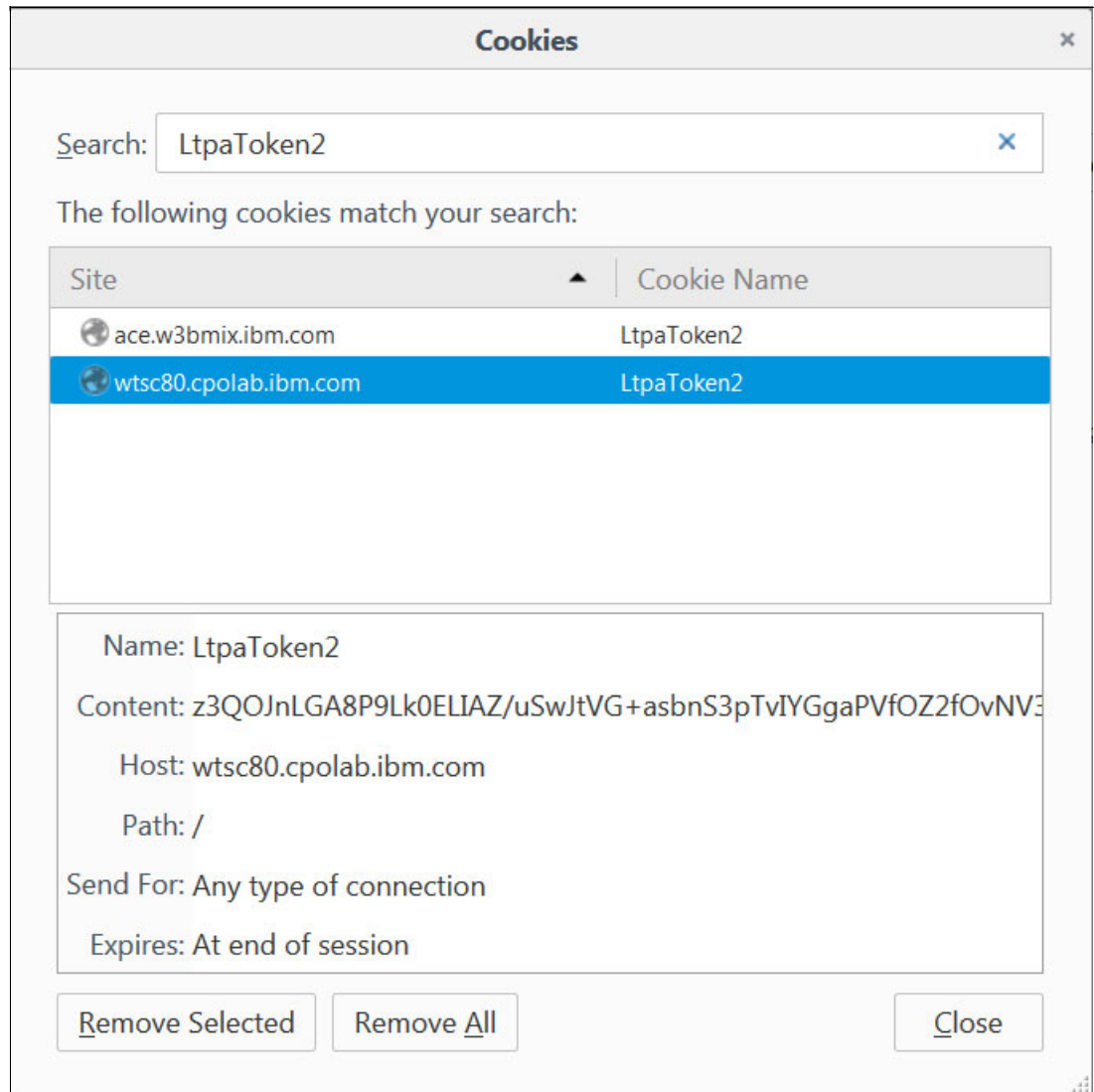


Figure 7-20 LTPA token as seen in the Firefox browser

- ▶ As described in 7.7, “JSON client code with cookie printer” on page 217, we provided a simple Java HTTP client that can be used with Java 8. No libraries are required to run this code, only base Java.

When you run this code, it prints any cookies that are returned by the server. The cookie output from our application when we ran it in our system is shown in Example 7-54. The cookie is encoded to prevent it from being human readable. Decrypting it requires the encryption keys from the LTPA keys file.

Example 7-54 LTPA token returned to our client by Liberty

```
LtpaToken2=kPnM6XfPMXLSRCLhRHaxK0K8h2MgrPPPhC00+hrYyIvLWntHoPI1++jdyREuSEbhHXyj+PeH
LF6iSVEEVBY583eDrOB3tFa20rB4iZA+iMjIL1Ue7/KLj1ioQ1JCcDAg3n0aB8wCRRBj7pKyDQ8hqFiAv
JKrnzcM2cQKox+lq1yn+1XB9gS6qPfd8a+3JXcHh5aBpjONbNxIhJwbDB0trKI9nwJhhdMT+k1frzUjywn
dI9inIjtf7aaVFYKd/2d1FIcpHRR0YPuKXdXqAOpWNnRMF1uKCMEarH56ESMoRx3aE11FpqGvmZPQb2yqv
Onu
```

7.6.2 Disabling SSO in Liberty

In some scenarios, you might not want to use LTPA or SSO in your system. If you are using security in your Liberty JVM server, LTPA is active by default. To turn off SSO, and thus in turn LTPA, you can add the `webAppSecurity` element to your Liberty JVM server's configuration file (`server.xml`). Add the element into your configuration with the element `singleSignOnEnabled` set to `false`.

The `webAppSecurity` configuration element that we used in our JVM server is shown in Example 7-55.

Example 7-55 webAppSecurity configuration with SSO disabled in server.xml

```
<webAppSecurity singleSignOnEnabled="false"/>
```

7.6.3 Requiring TLS when using SSO

You might want to add an extra layer of protection to your SSO configuration by requiring any connection with the server that uses LTPA (or other session token) to be made by using HTTPS rather than HTTP.

This protection can be enabled by adding the `webAppSecurity` configuration to your server's configuration file (`server.xml`). Add the `ssorequiresssl` attribute to the configuration and specify its value as `true`.

The `webAppSecurity` configuration element that we used to enforce TLS usage with SSO in our Liberty JVM server is shown in Example 7-56.

Example 7-56 webAppSecurity configuration with TLS for SSO in server.xml

```
<webAppSecurity ssorequiresssl="true"/>
```

7.7 JSON client code with cookie printer

The code that is shown in Example 7-57 is a simple Java 8 HTTP client that sends JSON data with HTTP basic authentication credentials to a URL by using HTTP POST. No other libraries are required to run this code, only base Java 8.

Example 7-57 JSON client code with cookie printer

```
import java.io.IOException;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Base64;
import java.util.Base64.Encoder;

/**
 * A very basic client for calling REST services in CICS
 * Liberty.
 *
 * This client is designed to run locally on a workstation.
 * Requires Java 8.
 */
```

```

* Note: ENCODING METHOD USED NOT SUITABLE FOR PRODUCTION
*       ENVIRONMENTS OR HIGHLY SENSITIVE INFORMATION
*
* @author Michael Jones/IBM-CICS (michaej8@uk.ibm.com)
*
*/

public class JSONClient {

    // Full path to our JAX-RS service
    String target =
        http://wtsc80:57080/stock/api/items;

    public static void main(String[] args) throws IOException {

        // Change the URL to point at your host and port
        URL url = new URL(target);

        // Create a very basic connection object which is configured
        // for HTTP POST
        HttpURLConnection conn =
            (HttpURLConnection)url.openConnection();
        conn.setRequestMethod("POST");

        // Encode the user's ID and password to send to the server
        // NOTE: Base64 is not a secure encoding. Not suitable for
        // production, only for personal testing
        Encoder encoder = Base64.getEncoder();
        String encodedCreds =
            encoder.encodeToString("WEBUSER:WEBUSER_PASSWORD".getBytes());
        conn.addRequestProperty("Authorization", "Basic " +
                                encodedCreds);

        // Set up some basic properties on the connection to prepare
        // it to send JSON
        conn.addRequestProperty("Content-Type", "application/json");
        conn.setDoOutput(true);
        conn.setDoInput(true);

        // The actual JSON data we want to send to the server. Modify
        // this as required.
        String sendJson = "{\"name\" : \"Golf Club\", \"stock\" : 1}";

        // Send the HTTP POST request to the server with the JSON data
        OutputStream os = conn.getOutputStream();
        os.write(sendJson.getBytes());

        // Go through and get cookies
        printCookies(conn);

        // Return the response to the command line
        System.out.println("Response code was: " +
                            conn.getResponseCode());
    }
}

```

```
public static void printCookies(HttpURLConnection conn) {  
  
    String cookieHeader = conn.getHeaderField("Set-Cookie");  
    if(cookieHeader == null) {  
        System.out.println("No cookies");  
        return;  
    }  
    System.out.println(cookieHeader);  
}  
}
```

You can change the target URL by changing the target field in the code. As shown in Example 7-57 on page 217, it is listing our machine's host name and ports and triggering the EJB shop sample's stock update service

You can also change the JSON data that is sent to the endpoint by updating the sendJson String object in the main method.

When the code runs, it sends the JSON request to endpoint that is listed. Then, it prints any cookies that are received in response from the server and the return code for the application.

The quickest way to run this code is by using Eclipse. Create a Java class in a new or existing project and paste in the code. Then, select **run** from the toolbar at the top of the GUI. No other set-up is required if Java 8 is installed.



Logging and monitoring

In this chapter, we describe how to configure the logging and monitoring destinations by using in the Liberty server environment in CICS.

We also describe where the logs and other diagnostics are written, and how to use the output to debug issues in practice. We also describe the key tools that can be used to monitor the JVM server status, including CICS statistics and monitoring, CICS Performance Analyzer, IBM Health Center, and Javametrics.

Finally, we review how to use CICS task runaway detection and use CICS policies to monitor transaction activity.

This chapter includes the following topics:

- ▶ 8.1, “Message and log files” on page 222
- ▶ 8.2, “Monitoring tools” on page 232

8.1 Message and log files

This section describes the output files that are related to a Liberty JVM server.

8.1.1 CICS logs

Several CICS logs are available for messages that are related to the JVM server.

MSGUSR DD file

Most errors that occur in a JVM server result in a message that is issued to the CICS MSGUSR DD log. This log is one of the first places to look for any CICS-related errors, warnings, or abends. A Java-based AJ05 abend in the CSMT queue that was produced by a `NullPointerException` in the Liberty CJSR transaction is shown in Figure 8-1.

```
DFHSJ0904 11/07/2017 11:01:57 SC8CICS2 CICSUSER ???? CJSR DFHSJTHP Exception 'java.lang.NullPointerException' occurred creating
object reference for class com.ibm.cics.wlp.impl.CICSHttpRunnable.
DFHAC2236 11/07/2017 11:01:57 SC8CICS2 Transaction CJSR abend AJ05 in program DFHSJTHP term ?????. Updates to local recoverable
resources will be backed out.
DFHSJ0904 11/07/2017 12:11:07 SC8CICS2 CICSUSER ???? CJSR DFHSJTHP Exception 'java.lang.NullPointerException' occurred creating
object reference for class com.ibm.cics.wlp.impl.CICSHttpRunnable.
DFHAC2236 11/07/2017 12:11:07 SC8CICS2 Transaction CJSR abend AJ05 in program DFHSJTHP term ?????. Updates to local recoverable
resources will be backed out.
```

Figure 8-1 AJ05 Abend message on MSGUSER sysout.

SYSPRINT and SYSOUT DD files

By default, the JVM `System.out` and `System.err` streams are sent to the Language Environment `stdout` and `stderr` streams. When run under batch MVS, the Language Environment `stdout` and `stderr` streams default to the `SYSPRINT` (`stdout`) and `SYSOUT` (`stderr`) destinations.

If either DD destinations are not defined in JCL, a dynamic `SYSnnnn` is created in the CICS `JOBLOG` for that stream.

To prevent a dynamic `SYSnnnn` being created, you can define the DD cards as shown in the following example:

```
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
```

Note: The `SYSOUT` DD is distinct from the `SYSOUT` parameter. The `SYSOUT =*` parameter is a redirection parameter and in this case, it sends output for the DD to the default output class on your JCL.

SYSOUT DD

The `SYSOUT` DD is a good place to look for any errors from the initialization of the JVM server. These messages are sent to the Language Environment `stderr` stream.

By introducing a JVM profile error in one of our CICS regions (`SC8CICS3`), a mis-configuration of the Java directory `JAVA_HOME=/usr/lpp/java/J8.0_64_SR44` produces a dynamically created `SYSOUT` file that is named `SYS00007`, as shown in Figure 8-2 on page 223.

Display Filter View Print Options Search Help									

SDSF JOB DATA SET DISPLAY - JOB SC8CICS3 (STC09328)						LINE 1-10 (10)			
COMMAND INPUT ==> █						SCROLL ==> CSR			
NP	DDNAME	StepName	ProcStep	DSID	Owner	C	Dest	Rec-Cnt	Page-Cnt
	JESMSG LG	JES2		2	CICSREGN	S		2	1
	JESJCL	JES2		3	CICSREGN	S		82	
	JESYSMSG	JES2		4	CICSREGN	S		2	
	DFHCXRF	SC8CICS3		105	CICSREGN	S		0	
	MSGUSR	SC8CICS3		109	CICSREGN	S		546	
	COUT	SC8CICS3		111	CICSREGN	S		0	
	CEEMSG	SC8CICS3		112	CICSREGN	S		0	
	CEEOUT	SC8CICS3		113	CICSREGN	S		0	
	CRPO	SC8CICS3		118	CICSREGN	S		0	
	SYS00007	SC8CICS3		119	CICSREGN	S	LOCAL	2	

Figure 8-2 SYSOUT generated by a profile error

When the JVM is initialized, the error is detected and the SYS0007 shows the error message. As when the Java library is wrong, the JVM fails to initialize. Figure 8-3 shows the contents of the SYS00007 file.

```
SDSF OUTPUT DISPLAY SC8CICS3 STC09331 DSID 119 LINE NOT PAGE MODE DATA
COMMAND INPUT ==> █ SCROLL ==> CSR
***** TOP OF DATA *****
2017/10/17 17:12:49.791525 ITS0JVM3 D YLE-Thread-0" YDFHSJSC" *Exc* : send_profile_message() - Problem encountered on line 47 of the
JVM profile ITS0JVM3.jvmprofile = Failed to open JAVA_HOME. EDC5129I No such file or directory.
***** BOTTOM OF DATA *****
```

Figure 8-3 Error showing the wrong Java library

8.1.2 Java logs

The JVM server provides the following specific log destinations:

► STDOUT

When the JVM server initializes, output from the System.out stream is written to the STDOUT destination that is set by the STDOUT option within the JVM server profile. If the file exists, the output is appended to the end of the file.

If the console log option is enabled, more information is recorded in this file. For more information, see 8.1.3, “Liberty server logs” on page 224. By default, this file is in the WORK_DIR/<applid>/<jvmserver> directory in zFS. In our scenario, the file was redirected to JES setting STDOUT=//DD:JVMOUT in the JVM profile.

► STDERR

The STDERR destination is the primary location to which Java exceptions and stack traces are written. These exceptions are typically a result of errors in Java applications or components. The location is set by using the STDERR option in the JVM server profile.

If the file exists, output is appended to the end of the file. By default, this file is in the WORK_DIR/<applid>/<jvmserver> directory in zFS. In our scenario, the file was redirected to JES setting STDOUT=//DD:JVMERR in the JVM profile.

Figure 8-4 shows the first part of a Java stack trace that was written to the stderr, which was caused by a `NullPointerException` in our restapp application.

```

  Display Filter View Print Options Search Help
-----
SDSF OUTPUT DISPLAY SC8CICS2 JOB02100 DSID 116 LINE 0 COLS 02- 161
COMMAND INPUT ==>
***** TOP OF DATA *****
ITS0JVM1: [err] 2017/11/28 12:04:13.271000 EST E [DFHSJTHP.TASK57.CJSA] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Error: handleThrowableErrors() -
java.lang.RuntimeException: AJ05
at com.ibm.cics.server.Wrapper.SetAbend(Native Method)
at com.ibm.cics.server.Wrapper.handleReturnCode(Wrapper.java:1238)
at com.ibm.cics.wlp.impl.CICSTaskWrapper.handleThrowableErrors(CICSTaskWrapper.java:839)
at com.ibm.cics.wlp.impl.CICSTaskWrapper.run(CICSTaskWrapper.java:395)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1160)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:635)
at com.ibm.cics.wlp.threading.CICSThread.run(CICSThread.java:245)
at com.ibm.cics.wlp.threading.CICSPooledThreadFactory.joinAsThreadInternal(CICSPooledThreadFactory.java:409)
at com.ibm.cics.wlp.threading.CICSPooledThreadFactory.joinAsThread(CICSPooledThreadFactory.java:335)
at com.ibm.cics.server.internal.ThreadJoiner.main(ThreadJoiner.java:106)
at sun.reflect.GeneratedMethodAccessor25.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:55)
at java.lang.reflect.Method.invoke(Method.java:508)
at com.ibm.cics.server.Wrapper.call_main(Wrapper.java:893)
at com.ibm.cics.server.Wrapper.callOSGiClass(Wrapper.java:2808)
at com.ibm.cics.server.Wrapper.invokeJvmServerOSGiClass(Wrapper.java:2675)
at com.ibm.cics.server.Wrapper.jvmServerOSGiEntry(Wrapper.java:2604)
at com.ibm.cics.osgi.impl.Controller.runService(Controller.java:1413)
at com.ibm.cics.osgi.impl.Controller.acceptRequest(Controller.java:322)
at sun.reflect.GeneratedMethodAccessor12.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:55)
at java.lang.reflect.Method.invoke(Method.java:508)
at com.ibm.cics.router.Router.route(Router.java:1315)
Caused by: java.lang.NullPointerException
at com.ibm.cicsdev.restapp.ErrorResource.getCICSInformation(ErrorResource.java:63)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:90)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:55)
at java.lang.reflect.Method.invoke(Method.java:508)

```

Figure 8-4 Abend AJ05 with `NullPointerException` written to stderr

8.1.3 Liberty server logs

The following log files are written to by the Liberty server:

- Liberty message log

The `messages.log` file contains all messages that are written or captured by the Liberty logging component. All messages that are written to this file contain more information, such as the message time stamp and the ID of the thread that wrote the message.

Each message features a unique message identifier, and includes an explanation of the problem and details of any action that you can take to resolve the problem. Liberty messages are logged from various sources, including application server components and applications.

The `messages.log` file is in the `logs` directory. You can control the size and number of log files by using the following system properties, which can be set in the JVM profile:

```

com.ibm.ws.logging.max.files
com.ibm.ws.logging.log.directory
com.ibm.ws.logging.max.file.name
com.ibm.ws.logging.max.file.size

```

This `messages.log` file can also be copied to JES by using an MSGLOG DD destination, as shown in the following example:

```
//MSGLOG DD SYSOUT=*
```

Figure 8-5 shows the Liberty messages that are related to UMASK value, included files, applications started, feature manager messages, and HTTP and HTTPS port allocation output to the MSGLOG DD file in the CICS job log.

```
SDSF ALTPHIT DISPLAY STARTS7 STC00250 DSID 003 TIME CHARS 'UMASK' FOLLO
COMMENC INPUT ---> SCROLL ---> CSR
I CWWKBD121I: The server process UMASK value is set to 0002.
[10/26/17 8:32:54:728 EDT] 0000002f com.ibm.us.config.xml.internal.XMLConfigParser
F CWWKGD0028N: Processing included configuration resource: /var/cicsts/SC8CICS7/wlp/servers/itsowlp1/installed0pps.xml
[10/26/17 8:32:54:730 EDT] 0000002f com.ibm.us.config.xml.internal.XMLConfigParser
F CWWKGD0028R: Processing included configuration resource: /var/cicsts/SC8CICS7/wlp/servers/itsowlp1/jms.xml
[10/26/17 8:32:55:068 EDT] 00000022 com.ibm.us.kernel.launch.internal.FrameworkManager
I CWWKE0002I: The kernel started after 1.18 seconds
[10/26/17 8:32:55:318 EDT] 00000030 com.ibm.us.kernel.feature.internal.FeatureManager
I CWWKF0007I: Feature update started.
[10/26/17 8:32:56:198 EDT] 0000002f com.ibm.us.security.ready.internal.SecurityReadyServiceImpl
I CWWKS0007I: The security service is starting...
[10/26/17 8:32:58:258 EDT] 0000002f container.security.feature.internal.FeatureAuthorizationTable
I CWWKS0120I: Authorization roles with id="com.ibm.cics.wlp.defaultapp" have been successfully processed.
[10/26/17 8:32:58:626 EDT] 0000002f com.ibm.us.app.manager.internal.monitor.DropInMonitor
F CWWKZ0058I: Monitoring dropins for applications.
I CWWK000219: TCP Channel defaultHttpEndpoint has been started and is now listening for requests on host wtc80.cpolab.ibm.com
(IPv4: 9.76.81.121) port 57080.
I CWWK000210: TCP Channel defaultHttpEndpoint ssl has been started and is now listening for requests on host wtc80.cpolab.ibm
(IPv4: 9.76.81.131) port 57443.
N CWWKT0016: Web application available (default host): http://wtc80.cpolab.ibm.com:57080/jmsweb/
[10/26/17 8:33:02:243 EDT] 0000008a com.ibm.ws.app.manager.AppMessageHelper
R CWWKZ0001: Application com.ibm.cicsdev.ngjms.cf.web started in 0.230 seconds.
R CWWKT0018: Web application available (default host): http://wtc80.cpolab.ibm.com:57080/com.ibm.cicsdev.restappext/
[10/26/17 8:33:02:508 EDT] 00000055 com.ibm.ws.app.manager.AppMessageHelper
R CWWKZ0001: Application com.ibm.cicsdev.restappext started in 0.462 seconds.
R CWWKFD012: The server installed the following features: [servlet 3.1, as: 1.0, jndi 1.0, jca 1.7, appSecurity 2.0, jaxrs 2.
, blueprint 1.0, wsqmsClient 2.0, xssSecurity 1.0, jsfBoundSecurity 1.0, jsf 2.0, cicsts:link 1.0, ejbLite 3.2, jsf 2.2, cicsts:
security-1.0, jsmp-1.0, el-3.0, jaxrsClient-2.0, jaxb-2.2, mdu-3.2, cicsts:defaultApp-1.0, cicsts:core-1.0, json-1.0, distributedMa
-1.0, web-1.0, websocket-1.1].
I CWWKS2060W: Cannot create the default credential for SNA authorization of unauthenticated users. All authorization checks f
r unauthenticated users will fail. The default credential could not be created due to the following error: CWWKS2007E: SAF Service
RASIF00 CREATE did not succeed because user CICSREGN has insufficient authority to access APPL-ID SC8CICS. SAF return code 0x000000
8, RACF return code 0x00000008, RACF reason code 0x00000023.
[10/26/17 11:54:35:793 EDT] 000000a3 com.ibm.us.security.saf.SAFServiceResult
E CWWKS2011E: SAF Service RACFOLTE FASTAUTH did not succeed because the resource profile SC8CICS.com.ibm.cics.wlp.defaultapp.
per in class EJBR0LE does not exist. SAF return code 0x00000004, RACF return code 0x00000004, RACF reason code 0x00000003.
[10/26/17 11:54:35:791 EDT] 000000a3 com.ibm.ws.webcontainer.security.WebAppSecurityCollaboratorImp
I CWWKS0104R: Authorization failed for user WEBUSER while invoking com.ibm.cics.wlp.defaultapp on /. The user is not granted
access to any of the required roles: [User, Administrator].
```

Figure 8-5 MSGLOG output

► Liberty trace

The trace.log file contains all trace entries that are written or captured by Liberty. This file is created only if you enable Liberty tracing (usually at the request of IBM level 2 support). Within the JVM profile, selectively enable this trace and specify the file name by using the following parameters:

```
com.ibm.ws.logging.trace.file.name=trace.log
com.ibm.ws.logging.trace.specification=SSLChannel=all:SSL=all
```

The format of the log detail level specification is <component> = <level> where <component> is the component for which to set a log detail level, and <level> is one of the valid logger levels (off, fatal, severe, warning, audit, info, config, detail, fine, finer, finest, or all). Separate multiple log detail level specifications by using colons (:).

By default, this file is in the Liberty logs directory along with the messages.log file.

► HTTP access log

The HTTP access log file contains a record of all inbound client requests that are handled by HTTP endpoints in Liberty. You can enable it in the Liberty server for each defined httpEndpoint by using the <httpAccessLogging> element. Example 8-1 on page 226 shows how it can be enabled in server.xml.

Example 8-1 Access logging configuration

```
<httpEndpoint id="defaultHttpEndpoint"
    host="wtsc80"
    httpPort="57080"
    httpsPort="57443"
    accessLoggingRef="accessLogging" />
<httpAccessLogging id="accessLogging"
    filepath="${server.output.dir}/logs/http_access.log"
    logFormat="%t %a %i %r %s %u %D %B" />
```

The logFormat attribute includes the following directives for a customized request:

- %t: Time and date
- %a: Remote IP address
- %i: Contents of the Header header-name in the request
- %r: First line of the request URL
- %s: HTTP status code
- %u: Remote user
- %D: Response time of the request in ms
- %B: Response size in Bytes, excluding headers

The information that is contained in the access log file after running our sample restapp application, the Javametrics dashboard, and the CICS defaultapp is shown in Figure 8-6.

File Edit Edit_Settings Menu Utilities Compilers Test Help									
VIEW /cicsts/SC8CICS7/wlp/servers/itsowlp1/logs/http_access.log					Columns 00001 00124				
Command ==>					Scroll ==> CSR				
017128	[16/Oct/2017:10:45:37 -0400]	9.27.167.16	- GET	/com.ibm.cicsdev.restapp/rest/cicsinfo	HTTP/1.1 200 - 13653 169				
017129	[16/Oct/2017:10:45:38 -0400]	9.27.167.16	- GET	/com.ibm.cicsdev.restapp/rest/cicsinfo	HTTP/1.1 200 - 10287 169				
017130	[16/Oct/2017:10:45:39 -0400]	9.27.167.16	- GET	/javametrics-dash/graphmetrics/images/maximize_24.png	HTTP/1.1 304 - 4080 0				
017131	[16/Oct/2017:10:47:05 -0400]	9.27.167.16	- GET	/com.ibm.cicsdev.restapp/rest/cicsinfo	HTTP/1.1 200 - 5826 169				
017132	[16/Oct/2017:10:47:06 -0400]	9.27.167.16	- GET	/com.ibm.cicsdev.restapp/rest/cicsinfo	HTTP/1.1 200 - 8507 169				
017133	[16/Oct/2017:10:47:10 -0400]	9.27.167.16	- GET	/com.ibm.cics.wlp.defaultapp/	HTTP/1.1 200 - 3864 3634				
017134	[16/Oct/2017:10:47:10 -0400]	9.27.167.16	- GET	/com.ibm.cics.wlp.defaultapp/images/logo.jpg	HTTP/1.1 304 - 3338 0				
017135	[16/Oct/2017:10:47:11 -0400]	9.27.167.16	- GET	/com.ibm.cics.wlp.defaultapp/	HTTP/1.1 200 - 4113 3634				
017136	[16/Oct/2017:10:47:11 -0400]	9.27.167.16	- GET	/com.ibm.cics.wlp.defaultapp/images/logo.jpg	HTTP/1.1 304 - 4219 0				
017137	[16/Oct/2017:13:31:22 -0400]	9.27.167.16	- GET	/com.ibm.cics.wlp.defaultapp/	HTTP/1.1 200 - 10629 3634				
017138	[16/Oct/2017:13:31:23 -0400]	9.27.167.16	- GET	/com.ibm.cics.wlp.defaultapp/images/logo.jpg	HTTP/1.1 304 - 3703 0				
017139	[16/Oct/2017:13:31:25 -0400]	9.27.167.16	- GET	/com.ibm.cicsdev.restapp/rest/cicsinfo	HTTP/1.1 200 - 15916 169				
017140	[16/Oct/2017:14:03:22 -0400]	9.27.167.16	- GET	/com.ibm.cics.wlp.defaultapp/	HTTP/1.1 401 - 4836 0				
017141	[16/Oct/2017:14:03:24 -0400]	9.27.167.16	- GET	/com.ibm.cics.wlp.defaultapp/	HTTP/1.1 200 - 7797 3634				
017142	[16/Oct/2017:14:03:24 -0400]	9.27.167.16	- GET	/favicon.ico	HTTP/1.1 404 - 1838 28987				
017143	[16/Oct/2017:14:04:21 -0400]	9.27.167.16	- GET	/com.ibm.cics.wlp.defaultapp/	HTTP/1.1 200 - 7000 3634				
017144	[16/Oct/2017:14:05:33 -0400]	9.27.167.16	- GET	/com.ibm.cics.wlp.defaultapp/	HTTP/1.1 200 - 5925 3634				
017145	[16/Oct/2017:14:05:34 -0400]	9.27.167.16	- GET	/com.ibm.cics.wlp.defaultapp/images/logo.jpg	HTTP/1.1 304 - 4293 0				
017146	[16/Oct/2017:14:07:40 -0400]	9.27.167.16	- GET	/com.ibm.cics.wlp.defaultapp/	HTTP/1.1 200 - 4962 3634				
017147	[16/Oct/2017:14:21:52 -0400]	9.27.245.24	- GET	/com.ibm.cicsdev.restappext/rest/taskInformation	HTTP/1.1 403 - 90660 0				
017148	[16/Oct/2017:14:21:52 -0400]	9.27.245.24	- GET	/favicon.ico	HTTP/1.1 404 - 12074 28987				
017149	[16/Oct/2017:14:22:14 -0400]	9.27.245.24	- GET	/com.ibm.cicsdev.restappext/rest/taskInformation	HTTP/1.1 200 - 400392 38				
017150	[16/Oct/2017:14:22:15 -0400]	9.27.245.24	- GET	/favicon.ico	HTTP/1.1 404 - 3584 28987				

Figure 8-6 HTTP access log

One line per HTTP request is available, which provides an easy way of determining the requests were received by Liberty and the HTTP status code was for each request.

► FFDC file

The First Failure Data Capture (FFDC) feature runs in the background and collects events and errors that occur during JVM runtime. The information that it collects are written to log files in the logs/ffdc directory.

Summary files are shown in Figure 8-7 on page 227. One entry for each exception occurred in the system, as shown in Figure 8-8 on page 227. The FFDC can be useful in your problem determination efforts. Also, they might be required by the Liberty server support team if you open a PMR.

```

File Edit Edit_Settings Menu Utilities Compilers Test Help

VIEW      /cicsts/SC8CICS7/wlp/servers/itsowlp1/logs/ffdc/exception_summary_17.10.16_14.35.08.0.log      Columns 00001 00124
Command ==> |
*****
***** Top of Data *****
000001
000002 Index Count Time of first Occurrence Time of last Occurrence Exception SourceId ProbeId
000003 -----
000004      0      1 10/16/17 14:35:08:663 EDT 10/16/17 14:35:08:663 EDT java.lang.NoClassDefFoundError com.ibm.ws.injectioneng
000005      - /var/cicsts/SC8CICS7/wlp/servers/itsowlp1/logs
000006      1      1 10/16/17 14:35:08:672 EDT 10/16/17 14:35:08:672 EDT java.lang.NoClassDefFoundError com.ibm.wsspi.injection
000007      - /var/cicsts/SC8CICS7/wlp/servers/itsowlp1/logs
000008 -----
***** Bottom of Data *****

```

Figure 8-7 FFDC exception summary file

```

File Edit Edit_Settings Menu Utilities Compilers Test Help

VIEW      /cicsts/SC8CICS7/wlp/servers/itsowlp1/logs/ffdc/ffdc_17.10.16_14.35.08.1.log      Columns 00001 00124
Command ==> |
000001 -----Start of DE processing----- = [10/16/17 14:35:08:673 EDT]
000002 Exception = java.lang.NoClassDefFoundError
000003 Source = com.ibm.wsspi.injectionengine.MethodMap.getAllDeclaredMethods
000004 probeid = 106
000005 Stack Dump = java.lang.NoClassDefFoundError: javax.jms.JMSRuntimeException
000006 at java.lang.J9VMInternals.prepareClassImpl(Native Method)
000007 at java.lang.J9VMInternals.prepare(J9VMInternals.java:291)
000008 at java.lang.Class.getDeclaredMethods(Class.java:1006)
000009 at com.ibm.wsspi.injectionengine.MethodMap.getMethods(MethodMap.java:149)
000010 at com.ibm.wsspi.injectionengine.MethodMap.getAllDeclaredMethods(MethodMap.java:85)
000011 at com.ibm.ws.injectionengine.InjectionProcessorManager.getAllDeclaredMethods(InjectionProcessorManager.java:429)
000012 at com.ibm.ws.injectionengine.InjectionProcessorManager.processAnnotations(InjectionProcessorManager.java:231)
000013 at com.ibm.ws.injectionengine.AbstractInjectionEngine.processInjectionMetaData(AbstractInjectionEngine.java:514)
000014 at com.ibm.ws.injectionengine.osgi.internal.OSGiInjectionEngineImpl.processInjectionMetaData(OSGiInjectionEngineImpl.java:2
000015 at com.ibm.ws.injectionengine.ReferenceContextImpl.processImpl(ReferenceContextImpl.java:595)
000016 at com.ibm.ws.injectionengine.ReferenceContextImpl.process(ReferenceContextImpl.java:290)
000017 at com.ibm.ws.injectionengine.osgi.internal.OSGiReferenceContextImpl.process(OSGiReferenceContextImpl.java:31)
000018 at com.ibm.ws.webcontainer.osgi.webapp.WebApp.commonInitializationStart(WebApp.java:255)
000019 at com.ibm.ws.webcontainer.webapp.WebApp.initialize(WebApp.java:1039)
000020 at com.ibm.ws.webcontainer.webapp.WebApp.initialize(WebApp.java:6595)
000021 at com.ibm.ws.webcontainer.osgi.DynamicVirtualHost.startWebApp(DynamicVirtualHost.java:468)
000022 at com.ibm.ws.webcontainer.osgi.DynamicVirtualHost.startWebApplication(DynamicVirtualHost.java:463)
000023 at com.ibm.ws.webcontainer.osgi.WebContainer.startWebApplication(WebContainer.java:1120)

```

Figure 8-8 FFDC Exception log

► Console log

The Liberty console log is not enabled by default in a CICS Liberty JVM server. It can be enabled by using the following system property:

```
com.ibm.ws.logging.console.log.level={INFO|AUDIT|WARNING|ERROR|OFF}
```

The ERROR level is the highest level of logging and causes all INFO, AUDIT WARNING, and ERROR messages to be written to the JVM server stdout destination. These messages are the same as the messages that are written to the `messages.log` file but do not include the time stamp and thread identity information.

When written to the stdout destination, the messages can be redirected to JES and prefixed with the identity of the JVM server by using the `IDENTITY_PREFIX` option in the JVM server profile, as shown in Figure 8-9.

```

Display Filter View Print Options Search Help
-----
SDSF OUTPUT DISPLAY SC8CICG3 STC09054 DSID 101 LINE 41 COLS 02- 133
COMMAND INPUT ==> | SCROLL ==> CSR
ITS0JVM3: YINFO " CWKKS0008I: The security service is ready.
ITS0JVM3: YINFO " CWWKZ0018I: Starting application javametrics-dash-1.0.1.
ITS0JVM3: YINFO " SESN8501I: The session manager did not find a persistent storage location; HttpSession objects will be stored i
ITS0JVM3: YINFO " SRVE0169I: Loading Web Module: com.ibm.cics.wlp.defaultapp.
ITS0JVM3: YINFO " SRVE0250I: Web Module com.ibm.cics.wlp.defaultapp has been bound to default_host.
ITS0JVM3: YAUDIT " CWWKT0016I: Web application available (default_host): http://wtsc80.cpolab.ibm.com:53080/com.ibm.cics.wlp.defau
ITS0JVM3: YINFO " SRVE0169I: Loading Web Module: javametrics.web.
ITS0JVM3: YINFO " SRVE0250I: Web Module javametrics.web has been bound to default_host.
ITS0JVM3: YAUDIT " CWWKT0016I: Web application available (default_host): http://wtsc80.cpolab.ibm.com:53080/javametrics-dash/
ITS0JVM3: YAUDIT " CWWKZ0001I: Application javametrics-dash-1.0.1 started in 0.220 seconds.
ITS0JVM3: YAUDIT " CWWKFO012I: The server installed the following features: Yjsp-2.3, servlet-3.1, ssl-1.0, cicsts:security-1.0, j
ITS0JVM3: YINFO " CWWKFO008I: Feature update completed in 4.651 seconds.
ITS0JVM3: YAUDIT " CWWKFO011I: The server itsowlp3 is ready to run a smarter planet.
ITS0JVM3: YWARNING " CWNEN0070W: The javax.annotation.Resource annotation class will not be recognized because it was loaded from th
ITS0JVM3: YINFO " SESN0176I: A new session context will be created for application key default_host/javametrics-dash
ITS0JVM3: YINFO " SESN0176I: A new session context will be created for application key default_host/com.ibm.cics.wlp.defaultapp
ITS0JVM3: YINFO " SRVE9103I: A configuration file for a web server plugin was automatically generated for this server at /var/cic
ITS0JVM3: YINFO " SESN0172I: The session manager is using the Java default SecureRandom implementation for session ID generation.
ITS0JVM3: YINFO " CWRLS0010I: Performing recovery processing for local WebSphere server (itsowlp3).
ITS0JVM3: YINFO " SESN0172I: The session manager is using the Java default SecureRandom implementation for session ID generation.
ITS0JVM3: YINFO " DYNAL056I: Dynamic Cache (object cache) initialized successfully.
ITS0JVM3: YINFO " CWWKH0046I: Adding a WebSocket ServerEndpoint with the following URI: /javametrics-socket
ITS0JVM3: YINFO " CWRLS0012I: All persistent services have been directed to perform recovery processing for this WebSphere server

```

Figure 8-9 Console log output with INFO level

8.1.4 JVM server trace output

In addition to the logging that is produced by the JVM server, CICS provides some standard trace points in the SJ (JVM server) and AP domains. These trace points trace the actions that CICS takes in setting up and managing JVM servers and running Java transactions.

The SJ component traces exceptions and processing in SJ domain to the internal trace table when set at level 2. SJ level 3 and beyond produces Java logging that is written to a trace file in zFS. The name and location of the trace file is determined by the `JVMTRACE` option in the JVM profile. If you want to capture the maximum amount of trace, setting the SJ component to ALL ensures that both CICS internal trace and the zFS trace files are produced.

You also can control the format of the trace by using the following JVM server system property (choose the option for your own purposes):

```
com.ibm.cics.jvmserver.trace.format={FULL|SHORT|ABBREV}
```

The default option is `SHORT` and this option is used when you send diagnostic information to IBM service.

Activating trace

You can modify and control CICS tracing by using the `CETR` transaction. Enter `CETR` at a CICS workstation and after you see the map that is shown in Figure 8-10 on page 229. Then, press `PF4` for trace components.

CETR		CICS Trace Control Facility		SC82 SC8CICS2
Type in your choices.				
Item	Choice	Possible choices		
Internal Trace Status	==> STARTED	STArtd, STOpped		
Internal Trace Table Size	==> 12288 K	16K - 1048576K		
Auxiliary Trace Status	==> STOPPED	STArtd, STOpped, Paused		
Auxiliary Trace Dataset	==> A	A, B		
Auxiliary Switch Status	==> NO	NO, NExt, All		
GTF Trace Status	==> STOPPED	STArtd, STOpped		
Master System Trace Flag	==> ON	ON, OFF		
Master User Trace Flag	==> ON	ON, OFF		
When finished, press ENTER.				
PF1=Help 3=Quit 4=Components 5=Ter/Trn 9=Error List				

Figure 8-10 CETR trace control facility transaction

On the first trace options page, set the trace values for AP domain to 1-2, and for SJ domain to ALL, as shown in Figure 8-11 and Figure 8-12.

CETR		Component Trace Options		SC82 SC8CIC
Over-type where required and press ENTER.				PAGE 1 OF 5
Component	Standard	Special		
AP	1-2	1-2		
AS	1	1-2		
BA	1	1-2		
BM	1	1		
BR	1	1-2		
CP	1	1-2		
DC	1	1		
DD	1	1-2		
DH	1	1-2		
DM	1	1-2		
DP	1	1-2		
DS	1	1-2		
DU	1	1-2		
EC	1	1-2		
EI	1	1-2		
EJ	1	1-2		
EM	1	1-2		
PF: 1=Help 3=Quit 7=Back 8=Forward 9=Messages ENTER=Change				

Figure 8-11 Setting AP domain trace level

CETR		Component Trace Options		SC82 SC8CIC
Over-type where required and press ENTER.				PAGE 4 OF 5
Component	Standard	Special		
SH	1	1-2		
SJ	ALL	1-2		
SM	1	1-2		
SO	1	1-2		
ST	1	1-2		
SZ	1	1		
TC	1	1-2		
TD	1	1		
TI	1	1-2		
TR	1	1-2		
TS	1	1-2		
UE	1	1		
US	1	1-2		
WB	1	1-2		
WU	1	1-2		
W2	1	1-2		
XM	1	1-2		
PF: 1=Help 3=Quit 7=Back 8=Forward 9=Messages ENTER=Change				

Figure 8-12 Setting SJ domain trace level

Then, press Enter and PF3 to return, and enter STARTED in the Auxiliary Trace Status field.

To demonstrate this trace, we called our restapp application taskInformation service, and used the following URL to show how the trace information is output for a CICS Liberty Java task:

wtsc80:52080/com.ibm.cicsdev.restappext/rest/taskInformation

The taskInformation service returned the following JSON response, which provided more information about the transaction ID, user ID, and task number:

```
{"transid":"CJSA","userid":"CICSUSER","tasknum":"339"}
```

The start of the JVM server trace file, which was in the zFS file /var/cicsts/SC8CICS2/ITS0JVM1/D20171129.T160838.dfhjvmtrc, is shown in Figure 8-13.

```
2017/11/29 16:10:37.291000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Event: processRunnable() - CICS Http runnable found: [Ljava.lang.
2017/11/29 16:10:37.291000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Event: processRunnable() - uri = /com.ibm.cicsdev.restappext/res
2017/11/29 16:10:37.291000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Event: processRunnable() - request = GET
2017/11/29 16:10:37.291000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Event: processRunnable() - url + query = http://wtsc80.cpolab.ib
2017/11/29 16:10:37.291000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Event: processRunnable() - client IP = 9.78.64.11
2017/11/29 16:10:37.291000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Event: processRunnable() - client Port = 64130
2017/11/29 16:10:37.291000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Event: processRunnable() - client IP Family = 300
2017/11/29 16:10:37.291000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Event: processRunnable() - server IP = 9.76.61.131
2017/11/29 16:10:37.291000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Event: processRunnable() - server Port = 52080
2017/11/29 16:10:37.292000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Event: processRunnable() - server IP Family = 300
2017/11/29 16:10:37.292000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Event: run() - CICS runnable - early binding enabled
2017/11/29 16:10:37.294000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Entry: buildTransaction() - tranid = "CJSA", userid = "
2017/11/29 16:10:37.295000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Entry/Exit: deferredSecurityPropagationAllowed() - true
2017/11/29 16:10:37.295000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Exit: buildTransaction() - true
2017/11/29 16:10:37.295000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.server] [Wrapper] @Entry: threadPoolBindDTC()
2017/11/29 16:10:37.295000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.server] [Task] @Entry: getTask()
2017/11/29 16:10:37.295000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.server] [Task] @Event: getTask() - Task is null
2017/11/29 16:10:37.295000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.server] [API] @Event: constructor() - bindTask=false,com.ibm.cics.server.Task@3a44061
2017/11/29 16:10:37.295000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.server] [Task] @Entry: constructor()
2017/11/29 16:10:37.295000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.server] [Task] @Event: constructor() - getCommonData returned
2017/11/29 16:10:37.295000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.server] [Task] @Exit: constructor()
2017/11/29 16:10:37.295000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.server] [Task] @Exit: getTask()
2017/11/29 16:10:37.295000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.server] [Wrapper] @Event: initOSG6IWrapper() - Task object = 339
2017/11/29 16:10:37.295000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.server] [Wrapper] @Event: addTask() - Task = 339
2017/11/29 16:10:37.295000 EST D [RUN_SERVICE_Thread-28] [com.ibm.cics.server] [Wrapper] @Exit: threadPoolBindDTC()
2017/11/29 16:10:37.295000 EST D [DFHSJTHP_TASK339.CJSA] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Event: renameThread() - Thread 'RUN_SERVICE_Thread-28' renamed a
2017/11/29 16:10:37.295000 EST D [DFHSJTHP_TASK339.CJSA] [com.ibm.cics.server] [Task] @Entry/Exit: getUserID()
2017/11/29 16:10:37.296000 EST D [DFHSJTHP_TASK339.CJSA] [com.ibm.cics.wlp.impl] [CICSTaskWrapper] @Event: setThreadInfo() - New TaskID = 339, New TranID = CJSA, In
2017/11/29 16:10:37.296000 EST D [DFHSJTHP_TASK339.CJSA] [com.ibm.cics.wlp.impl] [CICSHttpRunnable] @Entry/Exit: run()
```

Figure 8-13 JVM server zFS trace

In this trace output, we can see the URL, the HTTP request type of GET, and the client IP address. This output is followed by the call to `buildTransaction()`, which sets the transaction ID to CJSA, and then attaches the thread to the CICS task 339. Finally, the thread name is switched from the `RUN_SERVICE_Thread-28` to `DFHSJTHP_TASK339.CJSA`.

In Figure 8-14, we can see the CICS AUX trace for the SJ=ALL and AP=1-2 components, which are formatted in short format by using the DFHTU710 trace utility program, and filtered by using task number 339. This trace shows how the `buildTransaction()` method starts the task in CICS on the T8 TCB T802E and then starts to call the `ASSIGN` command that is driven from the `JCICS Task.getTask()` method call.

00339	T802E	XM	1102	XMAT	EXIT	BUILD_TRANSACTION/OK	TRANNUM(0000339C)	RET-94455B18	16:10:37.2950529472	00.0000003208	=001080=
00339	T802E	MN	0B0A	MNOD	ENTRY	GET_ODR	ODR_BUFFER(33F4C208 , 00000000 , 000001EC)	RET-94456226	16:10:37.2950543183	00.0000013710	=001081=
00339	T802E	MN	0B0B	MNOD	EXIT	GET_ODR/OK	ODR_BUFFER(33F4C208 , 000001EC , 000001EC) C_TRANSACTION_GROUP_ID	RET-94456226	16:10:37.2950545708	00.0000002524	=001082=
00339	T802E	SJ	0C01	SJTH	ENTRY	INCREMENT_USE_COUNT	JVMSEVER(ITS0JVM1)	RET-94456386	16:10:37.2950549414	00.0000003706	=001083=
00339	T802E	DD	0301	DDLO	ENTRY	LOCATE_DIRECTORY_TOKEN	(14E76E80) ENTRY_NAME(155EA1F5) DIRECTORY_NAME(JVMD) NAME(ITS0JVM1)	RET-9446A3C2	16:10:37.2950553168	00.0000003754	=001084=
00339	T802E	DD	0302	DDLO	EXIT	LOCATE/OK	DATA_TOKEN(15A46000 , 00000004)	RET-9446A3C2	16:10:37.2950554716	00.0000001547	=001085=
00339	T802E	SJ	0C02	SJTH	EXIT	INCREMENT_USE_COUNT/OK		RET-94456386	16:10:37.2951912617	00.0001357900	=001086=
00339	T802E	SJ	0B02	SJJS	EXIT	BUILD_TRANSACTION/OK	WEB_RESOURCES() URIMAP_PATH() TRACE_LEVEL(8)	RET-3582C763	16:10:37.2951925913	00.0000013295	=001087=
00339	T802E	KE	0201	KEDD	ENTRY	INQUIRE_GLOBAL_TRACE		RET-93F9E464	16:10:37.2952872788	00.0000946875	=001088=
00339	T802E	KE	0202	KEDD	EXIT	INQUIRE_GLOBAL_TRACE/OK	MASTER_TRACE_FLAG(ON) SYSTEM_TRACE_FLAG(ON)	RET-93F9E464	16:10:37.2952875473	00.0000002685	=001089=
00339	T802E	KE	0201	KEDD	ENTRY	INQUIRE_TASK_TRACE		RET-93F9E502	16:10:37.2952877822	00.0000002348	=001090=
00339	T802E	KE	0202	KEDD	EXIT	INQUIRE_TASK_TRACE/OK	TRACE_TYPE(STANDARD)	RET-93F9E502	16:10:37.2952879790	00.0000001967	=001091=
00339	T802E	KE	0201	KEDD	ENTRY	INQUIRE_DOMAIN_TRACE	DOMAIN_TOKEN(00000010)	RET-93F9E5C2	16:10:37.2952880722	00.0000000932	=001092=
00339	T802E	KE	0202	KEDD	EXIT	INQUIRE_DOMAIN_TRACE/OK	STANDARD_TRACE_FLAGS(F2FF7AC4) SPECIAL_TRACE_FLAGS(FFFFFFFE)	RET-93F9E5C2	16:10:37.2952882773	00.0000002050	=001093=
00339	T802E	KE	0201	KEDD	ENTRY	INQUIRE_DOMAIN_TRACE	DOMAIN_TOKEN(00000026)	RET-93F9E5C2	16:10:37.2952883662	00.0000000888	=001094=
00339	T802E	KE	0202	KEDD	EXIT	INQUIRE_DOMAIN_TRACE/OK	STANDARD_TRACE_FLAGS(80000000) SPECIAL_TRACE_FLAGS(C0000000)	RET-93F9E5C2	16:10:37.2952884199	00.0000000537	=001095=
00339	T802E	KE	0201	KEDD	ENTRY	INQUIRE_DOMAIN_TRACE	DOMAIN_TOKEN(00000028)	RET-93F9E5C2	16:10:37.2952884790	00.0000000590	=001096=
00339	T802E	KE	0202	KEDD	EXIT	INQUIRE_DOMAIN_TRACE/OK	STANDARD_TRACE_FLAGS(80000000) SPECIAL_TRACE_FLAGS(C0000000)	RET-93F9E5C2	16:10:37.2952886743	00.0000001953	=001097=
00339	T802E	KE	0201	KEDD	ENTRY	INQUIRE_DOMAIN_TRACE	DOMAIN_TOKEN(00000027)	RET-93F9E5C2	16:10:37.2952887353	00.0000000610	=001098=
00339	T802E	KE	0202	KEDD	EXIT	INQUIRE_DOMAIN_TRACE/OK	STANDARD_TRACE_FLAGS(80000000) SPECIAL_TRACE_FLAGS(C0000000)	RET-93F9E5C2	16:10:37.2952887724	00.0000000371	=001099=
00339	T802E	AP	21E0	JCICS	ENTRY	DTCTask_getCommonData		REQ-94718DEE	16:10:37.2954132680	00.0001244956	=001100=
00339	T802E	AP	00E1	EIP	ENTRY	ASSIGN	REQ(0004) FIELD-A(34005D10 ..) FIELD-B(08000208)	RET-B58C52B4	16:10:37.2954193647	00.0000060966	=001101=
00339	T802E	AP	00E1	EIP	EXIT	BOUNDARY(0280)	REQ(00F4) FIELD-A(00000000) FIELD-B(00000208)	RET-B58C52B4	16:10:37.2954239843	00.0000046196	=001102=
00339	T802E	AP	21E0	JCICS	ENTRY	DTCTask_ProcessEIB		RET-94718DEE	16:10:37.2954368486	00.0000128642	=001103=
00339	T802E	AP	21E0	JCICS	EXIT	DTCTask_ProcessEIB		RET-94718DEE	16:10:37.2954412368	00.0000043881	=001104=
00339	T802E	AP	21E0	JCICS	EXIT	DTCTask_getCommonData		RET-94718DEE	16:10:37.2954422119	00.0000009750	=001105=
00339	T802E	AP	21E0	JCICS	ENTRY	DTCTerminal_getUserId		RET-94718DEE	16:10:37.2964336958	00.0009914838	=001106=
00339	T802E	AP	00E1	EIP	ENTRY	ASSIGN	REQ(0004) FIELD-A(34005D10 ..) FIELD-B(08000208)	RET-B58C52B4	16:10:37.2964366367	00.0000029409	=001107=
00339	T802E	AP	00E1	EIP	EXIT	BOUNDARY(0280)		RET-94834ECE	16:10:37.2964473295	00.0000106928	=001108=
00339	T802E	US	0401	USXM	ENTRY	INQUIRE_TRANSACTION_USER		RET-94834ECE	16:10:37.2964476347	00.0000003051	=001109=
00339	T802E	US	0402	USXM	EXIT	INQUIRE_TRANSACTION_USER/OK	USERID_LENGTH(8) USERID(CICSUSER) CURRENT_GROUPID_LENGTH(0)	RET-94834ECE	16:10:37.2964480126	00.0000003779	=001110=
00339	T802E	US	0402	USXM	EXIT	CURRENT_GROUPID() NATIONAL_LANGUAGE() USERNAME() OPERATOR_IDENT() OPERATOR_PRIORITY(0)		RET-94834ECE	16:10:37.2964480126	00.0000003779	=001110=
00339	T802E	AP	00E1	EIP	EXIT	ASSIGN OK	REQ(00F4) FIELD-A(00000000) FIELD-B(00000208)	RET-B58C52B4	16:10:37.2964480126	00.0000003779	=001110=
00339	T802E	AP	21E0	JCICS	EXIT	DTCTerminal_getUserId	C_USERID(CICSUSER)	RET-94718DEE	16:10:37.2964511660	00.0000031533	=001111=

Figure 8-14 CICS AUX trace

Stopping trace

To stop CICS writing an internal trace to the auxiliary trace dataset, you can set Auxiliary Trace Status to OFF in the CETR transaction.

To stop JVM server tracing, the values for the AP and SJ domains must be back to 1 in the CETR transaction because this value is not affected by the AUX or SYSTEM trace settings.

8.2 Monitoring tools

In this section, we describe several methods that can be used to monitor CICS JVM server activity, including CICS statistics and CICS performance class data. These methods can be extracted from system management facilities (SMF) 110 records.

8.2.1 CICS statistics records

CICS collects statistics for the JVM servers and JVM programs. You can use these statistics to manage and tune the Java workloads that are running in your CICS region. The JVM server statistics provide information about the JVM and activity within a particular JVM server. The JVM program statistics provide information about the use of CICS Java programs. In the context of a Liberty JVM server, these programs are the only Java programs that are started by using the Link to Liberty function (for more information, see Chapter 8, “Logging and monitoring” on page 221).

To view the statics online, use the sample STAT transaction that prints the statistics to the CICS log by using the DFH0STAT program. You also can develop your own reporting application by using the **EXEC CICS EXTRACT STATISTICS** command. If you use the CICS Explorer, the JVM Servers view also displays all the same statistics fields.

For batch reporting, the sample utility program DFHSTUP provides a simple method to report on any type of CICS statistics. A sample JCL for running this program to report on our JVM server is shown in Example 8-2.

Example 8-2 DFHSTUP sample JCL

```
//JOB ...
//STUP1 EXEC PGM=DFHSTUP,REGION=0M
//STEPLIB DD DISP=SHR,DSN=CICSTS54.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=CICSTS54.CICS.SDFHAUTH
//DFHSTATS DD DISP=SHR,DSN=CICSUSER.CICS.SMF110(0)
//SYSIN DD *
SELECT APPLID=(SC8CICS7)
COLLECTION TYPE=ALL
SELECT TYPE=(JVMSERVER)
DATE START=10/06/2017,STOP=10/06/2017
TIME START=00.00.00,STOP=23.00.00,ELAPSED
/*
//DFHSTWRK DD UNIT=SYSDA,SPACE=(CYL,(8,4))
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(4))
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(4))
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(4))
//SORTWK04 DD UNIT=SYSDA,SPACE=(CYL,(4))
//DFHPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSABEND DD DUMMY
```

The output from DFHSTUP regarding our JVMSERVER ITSOJVM1 is shown in Figure 8-15 on page 233. The statistic period is set for one-hour intervals (for 10/19/2017 from 16:00:00 to 17:00:00).

```

COMMAND INPUT ==> █ SCROLL ==> CSR
JVMSEVER Name . . . . . : ITS0JVM1
JVMSEVER JVM profile name . . . : ITS0JVM1
JVMSEVER LE runtime options . . . : DFHAXR0
JVMSEVER use count. . . . . : 1466
JVMSEVER thread limit . . . . . : 50
JVMSEVER current threads. . . . . : 13
JVMSEVER peak threads . . . . . : 27
JVMSEVER thread limit waits . . . : 0
JVMSEVER thread limit wait time . : 00:00:00.0000
JVMSEVER current thread waits . . : 0
JVMSEVER peak thread limit waits . : 0
JVMSEVER system thread use count. : 7
JVMSEVER system thread waits. . . : 0
JVMSEVER system thread wait time. : 00:00:00.0000
JVMSEVER current sys-thread waits.: 0
JVMSEVER peak system thread waits : 0
JVMSEVER state. . . . . : Enabled
JVMSEVER JVM creation time. . . . : 10/19/2017 11:18:11.8937
JVMSEVER current heap size . . . . : 97413368
JVMSEVER initial heap size . . . . : 128M
JVMSEVER maximum heap size . . . . : 256M
JVMSEVER peak heap size . . . . . : 131M
JVMSEVER heap occupancy . . . . . : 92092368
JVMSEVER Garbage Collection policy: -Xgcpolicy:gencon
JVMSEVER major GC collections. . . : 1
JVMSEVER elapsed time in major GC.: 7
JVMSEVER major GC heap freed . . . : 2727624
JVMSEVER minor GC collections. . . : 91
JVMSEVER elapsed time in minor GC.: 481
JVMSEVER minor GC heap freed . . . : 2712M
-----

```

Figure 8-15 JVMSEVER DFHSTUP output

The report shows the following information:

- ▶ Our Liberty JVM server was used 1466 times, which equates to all the HTTP and JMS MDB requests that were received.
- ▶ The thread limit of the JVM server was set to a maximum of 50.
- ▶ There are 13 active threads, which are the threads available in the pool.
- ▶ The peak thread usage was 27 threads.
- ▶ No requests are needed to wait for a thread; therefore, the limit of 50 threads appears to be fine for our JVM server.
- ▶ Seven calls were made to the system thread, most of which were caused by the use of **CEMT SET JVMSEVER** commands.
- ▶ The heap size is 97 MB out of a maximum of 256 MB.
- ▶ The heap size after the last garbage collection (GC) was 92 MB. This result is known as the occupancy and often remains stable in a well-tuned system.
- ▶ The GC policy in use is gencon. This policy splits the Java heap into two areas, the new or nursery area for short lived objects, and the old or tenured area for long-lived objects.
- ▶ One major GC event (for the tenured area) and 91 minor GC events for the short-lived objects were observed in the nursery area.

This report can be generated daily by using a statistics period of one hour to track the performance of the JVM server over time.

Tip: Another useful metric is total GC heap freed divided by the number of requests. This formula gives an average garbage per request figure, which is useful to track on across application changes.

For more information about how to use the DFHSTUP and DFH0STAT reporting programs, [see IBM Knowledge Center](#).

8.2.2 CICS performance records

CICS provides the following programs for processing any CICS monitoring data that is written to SMF data sets:

► DFHMNDUP

This utility program generates a performance dictionary record in a sequential data set for use with monitoring data that is extracted from SMF data sets. The JCL for running this program is shown in Example 8-3.

Example 8-3 DFHMNDUP sample JCL

```
//JOB ...
//MNDUP EXEC PGM=DFHMNDUP
//STEPLIB DD DSN=CICSTS54.CICS.SDFHLOAD,DISP=SHR
//SYSUT4 DD DSN=CICSUSER.CICS.MNDUPREC,DISP=(NEW,CATLG),
//          UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
MCT=NO
SYSID=SC80
GAPPLID=SC8CICS7
//*
```

► DFH\$MOLS

This print program is for CICS monitoring data. DFH\$MOLS is a sample program that you can modify or adapt to your own purposes. The JCL for running this program is shown in Example 8-4.

Example 8-4 DFH\$MOLS sample JCL

```
//JOB ...
//EXPAMND EXEC PGM=DFH$MOLS
//STEPLIB DD DSN=CICSTS54.CICS.SDFHLOAD,DISP=SHR
//INPUT DD DISP=OLD,DSN=CICSUSER.CICS.MNDUPREC
// DD DISP=SHR,DSN=CICSUSER.CICS.SMF110(0)
//SYSPRINT DD SYSOUT=A
//SYSOUT DD SYSOUT=A
//SORTDIAG DD SYSOUT=A
//SYSIN DD *
SELECT APPLID=SC8CICS7
SELECT TRANID=CJSA,CJSU,JDB2,JRES,JJMS

PRINT PER
RESOURCE ALL
//*
```

The output report from DFH\$MOLS that is shown in Figure 8-16 on page 236 displays the following information for a single CICS task from an HTTP request to our Java web application:

- ▶ The transaction name (TRAN), in this case it is CJSA, which is the default JVM server request processor transaction.
- ▶ The start and stop time stamps for the CICS task (START and STOP).
- ▶ The number of performance class records that is recorded is 1 (PERRCENT).
- ▶ The CICS task number (TRANNUM) and transaction priority (TRANPRI).
- ▶ The IP address (9.27.245.24) of the browser (CLIPADDR).
- ▶ The dispatch time (USRDISPT) across all CICS TCBs, which is 1081 ms. This dispatch time equates approximately to the response time.
- ▶ The CPU usage timers, including:
 - The total CPU time on all types of CPU (USRCPUT), which is 810 ms.
 - The amount of CPU that is used by the task on the Java T8 TCB (T8CPUT), which is also 810 ms.
 - The amount of CPU that is used by the task on the QR TCB (QRCPUT), which is minimal at 5 μ s.
 - The time that is spent waiting for dispatch on to the QR TCB (QRMODDLY), which is insignificant at 4 μ s.
 - The CPU time on a general-purpose CPU (CPUTONCP), which is 3.895 ms.
 - CPU time that was eligible for zIIP offload, but ran on a general-purpose CPU (OFFLCPUT), which is 3.160 ms.

zIIP analysis: From the metrics that are shown in Figure 8-16 on page 236, we can calculate the CPU time that is used on a zIIP engine for our task is USRCPUT-CPUTONCP, which is 806 ms.

We can also calculate the total time spent zIIP eligible, which is the amount of time that is spent on a zIIP engine (806 ms as shown in Figure 8-16 on page 236), plus the extra 3.16 ms, which was zIIP eligible but ran on a general-purpose CP (OFFLCPUT), which makes 809 ms.

Therefore, the zIIP offload eligible fraction is the amount of zIIP-eligible CPU time (809 ms) divided by the total CPU time (810 ms), or 99.9% for this task.

DFHTASK	C001	TRAN	C3D1E2C1	CJSA					
...									
DFHCICS	C089	USERID	C3C9C3E2 E4E2C5D9	CICSUSER					
DFHTASK	C004	TTY	E4400000	U					
DFHCICS	T005	START	D34F7880E24FC84A	2017/10/19 14:34:01.433340					
DFHCICS	T006	STOP	D34F7881EA535E52	2017/10/19 14:34:02.514741					
DFHTASK	P031	TRANNUM	0016454C	16454					
DFHTASK	A109	TRANPRI	00000001	1					
...									
DFHPRG	C071	PGMNAME	C4C6C8E2 D1E3C8D7	DFHSJTHP					
DFHTASK	C097	NETUOWPX	E4E2C9C2 D4E2C34B E2C3F8C3 C9C3E2F7 00000000	USIBMSC.SC8CICS7					
DFHTASK	C098	NETUOWSX	4F7880E252DE0005						
...									
DFHCICS	A131	PERRECNT	00000001	1					
DFHTASK	T132	RMUOWID	D34F7881EB1D696	2017/10/19 14:34:02.467101					
...									
DFHTASK	A164	TRANFLAG	8000800015A00000						
...									
DFHSOCK	C318	CLIPADDR	F94BF2F7 4BF2F4F5 4BF2F440 40404040 40404040 9.27.245.24						
		+X0014	40404040 40404040 40404040 40404040 40404040						
DFHTASK	C082	TRNGRPID	1910E4E2C9C2D4E2. . .						
DFHTASK	S007	USRDISP	000000010802B56C00000003	000 00:00:01.081387					3
DFHTASK	S008	USRCPUT	00000000C5DD866400000003	000 00:00:00.810459					3
DFHTASK	S436	CPUTONCP	0000000000F3712700000003	000 00:00:00.003895					3
DFHTASK	S437	OFFLCPUT	0000000000C58FDB00000003	000 00:00:00.003160					3
DFHTASK	S014	SUSPTIME	000000000000E09C00000003	000 00:00:00.000014					3
DFHTASK	S102	DISPWT	0000000000006A5400000002	000 00:00:00.000006					2
DFHTASK	S255	QRDISPT	00000000000055CA00000001	000 00:00:00.000005					1
DFHTASK	S256	QRCPUT	00000000000053C200000001	000 00:00:00.000005					1
...									
DFHTASK	S262	KY8DISPT	0000000108025FA200000002	000 00:00:01.081381					2
DFHTASK	S263	KY8CPUT	00000000C5DD62A200000002	000 00:00:00.810454					2
...									
DFHTASK	S400	T8CPUT	00000000C5DD62A200000002	000 00:00:00.810454					2
DFHTASK	S249	QRMODDLY	0000000000004FCC00000001	000 00:00:00.000004					1
...									
DFHTASK	S247	DSCHMDLY	000000000000703400000002	000 00:00:00.000007					2
...									
DFHTASK	S125	DSPDELAY	000000000000706800000001	000 00:00:00.000007					1
...									
DFHTASK	S253	JVMTIME	0000000107F80ECE00000001	000 00:00:01.081216					1

Figure 8-16 DFH\$MOLS output for CJSA transaction

All of these fields are taken from the SMF 110.1 record (subtype 1). For more information about programming the structure of CICS SMF type 110, and how the monitoring data is packaged in the SMF records, [see IBM Knowledge Center](#).

8.2.3 CICS Performance Analyzer

CICS Performance Analyzer (CICS PA) is a reporting tool that provides information about the performance of your CICS systems and applications. It helps you tune, manage, and plan your CICS systems effectively. By using the SMF records from our current scenario, CICS PA can be used to easily generate more detailed performance reports.

A CICS PA report form that includes the fields that are related to JVM usage and CPU time is shown in Figure 8-17 on page 237. The report definition also allows transaction selection. By using the edit form facility, you can select the fields of your choice to create your own report.

```

File Edit Confirm Upgrade Options Help

EDIT LIST Report Form - PERFJVM7      Row 1 of 399 More: >
Command ==> █                          Scroll ==> CSR

Description . . . List JVM cpu consumption      Version (VRM): 710

Selection Criteria:
_ Performance *                               Page width . . 132

Field
/ Name +   Type   Fn   Description
---
TRAN       _____ Transaction identifier
TASKNO     _____ Transaction identification number
RESPONSE   _____ Transaction response time
DISPATCH  TIME    _____ Dispatch time
CPU        TIME    _____ CPU time
JVMSUSP    TIME    _____ JVM suspend time
JVMTHDWT   TIME    _____ JVM server thread wait time
JVMTIME    TIME    _____ JVM elapsed time
CPUONSP     _____ CPU time on Specialty Processor
CPUONCP    TIME    _____ CPU time on standard CP
CPUONCPE   TIME    _____ Offload eligible CPU time on standard CP
CPUONCPN   _____ CPU time on standard CP not offload eligible
CPUISSPE   _____ CPU time that is offload eligible
CPUSU      _____ CPU Service Units
ABCODEC    _____ Current ABEND code

```

Figure 8-17 CICS performance analyzer report form

The fields that we selected in our example report show the following information:

- ▶ TRAN: Transaction identifier and task number.
- ▶ RESPONSE: Transaction response time.
- ▶ DISPATCH: Total user task dispatch time.
- ▶ CPU: CPU time across all CICS TCBs.
- ▶ JVMSUSP: Time in which the user task was suspended.
- ▶ JVMTHDWT: Time waiting for a JVM server thread.
- ▶ JVMTIME: Elapsed time that is spent in the CICS JVM by the user task.
- ▶ CPUONSP: CPU time that was offloaded to zIIP speciality processors.
- ▶ CPUONCP: CPU time on general-purpose CPU.
- ▶ CPUONCPE: CPU time on general-purpose CPU that can be offloaded but was not.
- ▶ CUPONCPN: CPU time on general-purpose CPU that was not eligible for offload.
- ▶ CPUISSPE: CPU time that is converted to Service Units.
- ▶ ABCODEC: Task abend code, if any.

After the job is submitted, the CICS PA report shows results for selected transactions, as shown in Figure 8-18 on page 238.

Note: The use of CICS PA for performance reporting can be easier than the use of DFH\$MOLS because several other fields are automatically calculated, such as CPUONSP.

V5R4M0		CICS Performance Analyzer Performance List														
LIST0001 Printed at 17:36:06 10/19/2017				Data from 10:34:02 10/19/2017								APPLID SC8CICS7		Page		
Tran	TaskNo	Response Time	Dispatch Time	User Time	CPU Time	JVM Time	Susp Time	JVMThdWt Time	JVM Time	Elap Time	CPUonSP Time	CPUonCP Time	CPUonCpe Time	CPUonCPn Time	CPUisSPe Time	SrvcUnit
CJSA	16454	1.0814	1.0814	.8105	.0000	.0000	.0000	.0000	1.0812	.8066	.0039	.0032	.0007	.0007	.8097	716445.7
CJSA	16455	.0068	.0068	.0053	.0000	.0000	.0000	.0000	.0067	.0051	.0002	.0000	.0002	.0001	.0051	4641.865
CJSA	16456	.0056	.0056	.0038	.0000	.0000	.0000	.0000	.0055	.0036	.0002	.0000	.0002	.0002	.0036	3391.259
JMTR	16506	.0007	.0006	.0006	.0000	.0000	.0000	.0000	.0005	.0004	.0002	.0000	.0002	.0000	.0004	526.0254
JRST	16507	.0598	.0598	.0546	.0000	.0000	.0000	.0000	.0596	.0544	.0002	.0000	.0002	.0000	.0544	48283.29
JRST	16508	.0020	.0019	.0017	.0000	.0000	.0000	.0000	.0018	.0016	.0002	.0000	.0002	.0000	.0016	1545.356
JRST	16509	.0025	.0024	.0021	.0000	.0000	.0000	.0000	.0023	.0021	.0000	.0000	.0000	.0000	.0021	1887.721
JRST	16510	.0020	.0019	.0018	.0000	.0000	.0000	.0000	.0019	.0018	.0000	.0000	.0000	.0000	.0018	1587.073
JRST	16823	.0018	.0018	.0015	.0000	.0000	.0000	.0000	.0017	.0014	.0001	.0000	.0001	.0001	.0014	1362.337
JRST	16824	.0023	.0023	.0022	.0000	.0000	.0000	.0000	.0022	.0020	.0003	.0000	.0003	.0000	.0020	1957.776
JDFA	16825	.0026	.0026	.0024	.0000	.0000	.0000	.0000	.0025	.0021	.0003	.0000	.0003	.0000	.0021	2142.499
JDFA	16826	.0021	.0021	.0020	.0000	.0000	.0000	.0000	.0020	.0018	.0002	.0000	.0002	.0000	.0018	1750.597
JDFA	16827	.0022	.0022	.0013	.0000	.0000	.0000	.0000	.0021	.0013	.0000	.0000	.0000	.0000	.0013	1141.392
JDFA	16828	.0016	.0016	.0015	.0000	.0000	.0000	.0000	.0015	.0015	.0000	.0000	.0000	.0000	.0015	1344.077
JJMS	16831	.0038	.0035	.0028	.0000	.0000	.0000	.0000	.0034	.0025	.0003	.0000	.0003	.0000	.0025	2497.473
JJMS	16832	.0274	.0273	.0263	.0000	.0000	.0000	.0000	.0272	.0259	.0004	.0001	.0002	.0001	.0261	23245.18
JJMS	16834	.0093	.0093	.0084	.0000	.0000	.0000	.0000	.0091	.0078	.0006	.0002	.0004	.0000	.0080	7451.491
CJSA	16835	.0632	.0631	.0592	.0000	.0000	.0000	.0000	.0629	.0585	.0007	.0000	.0007	.0000	.0585	52334.97
JJMS	16836	.2422	.2422	.2260	.0000	.0000	.0000	.0000	.2420	.2258	.0002	.0000	.0001	.0001	.2258	199774.8
CJSA	16837	.0264	.0260	.0214	.0004	.0000	.0000	.0000	.0262	.0203	.0011	.0002	.0010	.0000	.0204	18930.26
CJSA	16841	.0039	.0036	.0029	.0002	.0000	.0000	.0000	.0036	.0018	.0011	.0000	.0011	.0000	.0018	2520.393
JURI	16844	.0032	.0031	.0028	.0000	.0000	.0000	.0000	.0030	.0021	.0007	.0000	.0007	.0000	.0021	2471.518
JMTR	51	.0457	.0457	.0400	.0000	.0000	.0000	.0000	.0445	.0393	.0007	.0000	.0007	.0000	.0393	35354.03

Figure 8-18 CICS PA performance list

For more information about the CICS PA fields and how to correlate these fields with the CICS SMF 110 performance class data, [see IBM Knowledge Center](#).

8.2.4 CICS Explorer

CICS Explorer is a management tool for developers and systems administrators. It offers a graphical interface for managing one or more CICS environments. It can be used for most systems management tasks, including viewing CICS system initialization (SIT) parameters, managing resource definitions, and building and deploying CICS bundle projects.

CICS Explorer is required if you deploy applications by using CICS bundles, including Java applications that are deployed into Liberty.

In our scenario, CICS Explorer is used to monitor the JVM servers, bundles and bundleparts, URI maps, and other related resources. A sample CICS Explorer JVM server operations view, which includes the JVM server statistics records, is shown in Figure 8-19 on page 239.

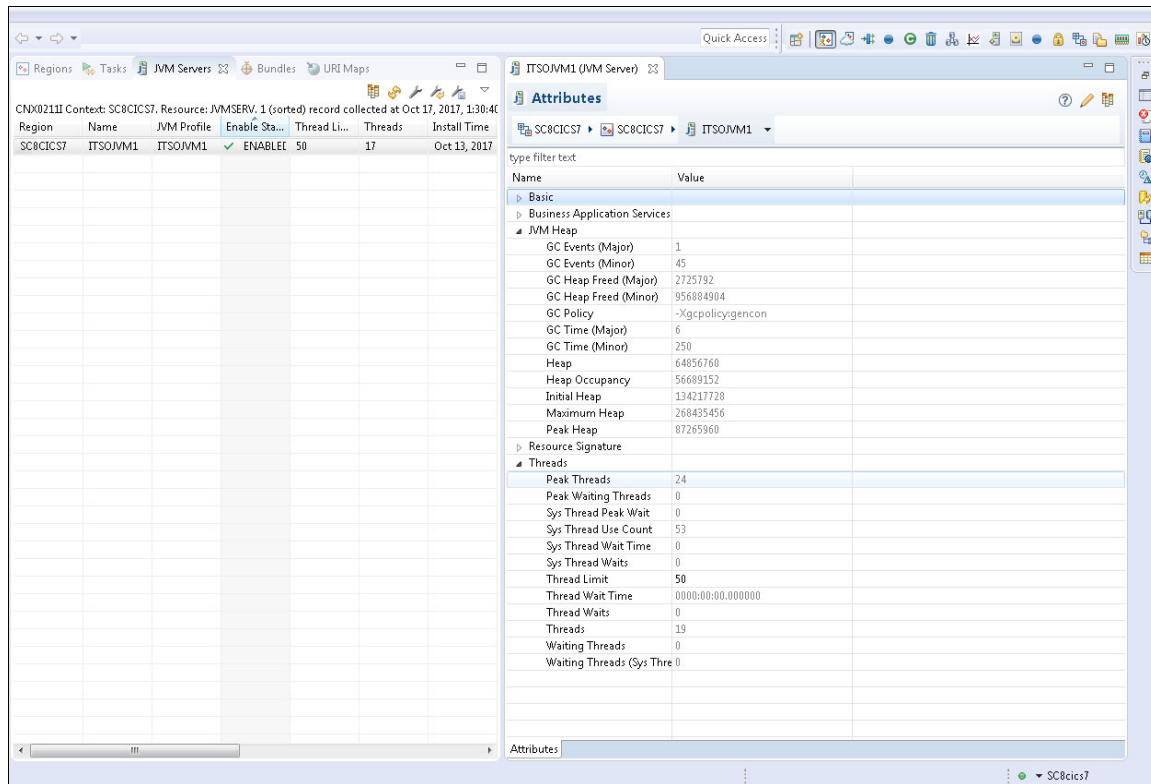


Figure 8-19 CICS Explorer JVM Server view

8.2.5 IBM Health Center

IBM Health Center is a diagnostic tool for monitoring the status of a Java virtual machine (JVM) and can be used with any IBM JVM, including CICS JVM servers. It features low runtime overhead and provides a highly customizable client as part of the IBM Support Assistant (ISA) workbench. Because the Health Center client is Eclipse based, it can also be installed into other Eclipse environments, such as the IBM CICS Explorer or z/OS Explorer.

The first step is to install the Health Center plug-in into your Eclipse client. Complete the following steps to add a software repository to your Eclipse:

1. In the CICS Explorer, click **Help** → **Install New Software**.
2. Click **Add** to add a software repository and provide the following location, as shown in Figure 8-20:
 - Information Name: IBM Health Center site
 - Location: <http://public.dhe.ibm.com/software/websphere/runtimes/tools/healthcenter/>

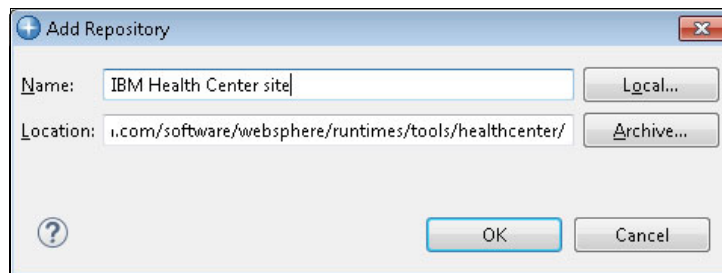


Figure 8-20 Adding the Health Center repository

3. Click **Next** in the Installation window (see Figure 8-21) to confirm the items to install.

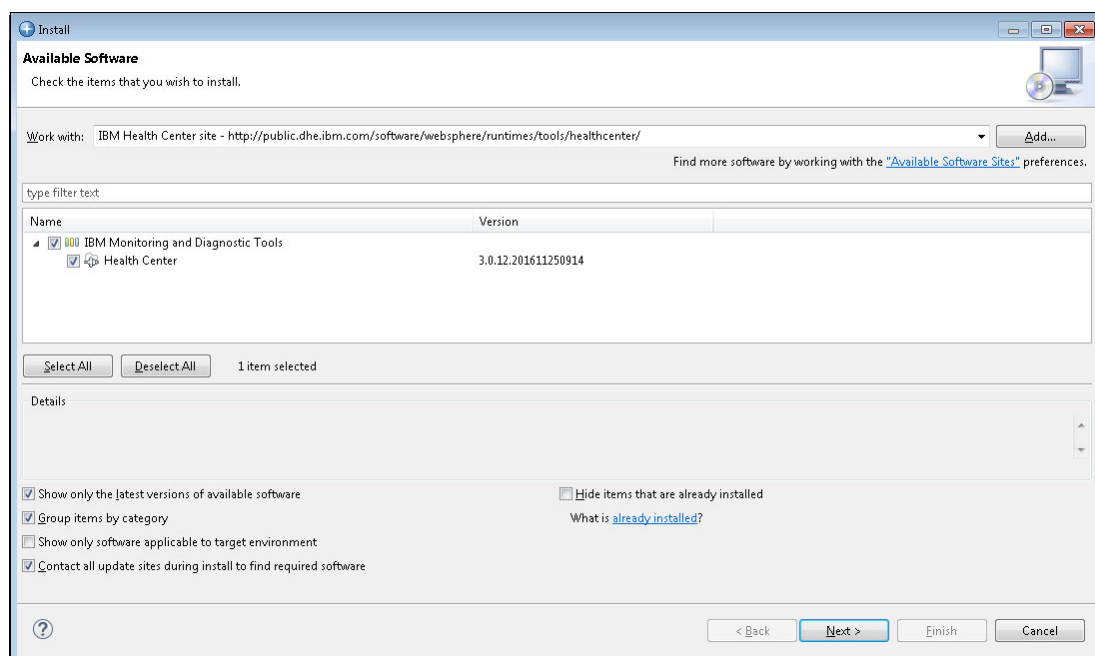


Figure 8-21 Health Center Installation window

4. Review the license information and click **Next**.
5. Click **OK** to confirm the installation of unsigned content, and then, click **Yes** to restart the Eclipse when prompted.

Enabling the JVM server for monitoring

To enable a JVM for monitoring, it is necessary to add a few JVM arguments to enable the Health Center agent to listen on the required TCP/IP port. When a CICS JVM server is used, the JVM arguments are set in the JVM profile, which is in the UNIX System Services directory that is specified by the JVMPROFILEDIR SIT parameter.

Add the following `-Xhealthcenter` JVM argument in your JVM profile. This argument enables the Health Center agent to collect monitoring data and specifies the listening port. We specified the port number 57108, as shown in the following example:

```
-Xhealthcenter:port=57108
```

The JVM must be disabled or enabled to implement this change. After the JVM server is started, the Health Center agent initializes and immediately starts buffering monitoring data ready for collection by the Health Center client. At this stage, you can check that the Health Center agent is listening on the TCP/IP port by using the following MVS TSO command:

```
NETSTAT (PORT 57108)
```

Note: Only one Eclipse client session can be connected to an IBM Health Center agent at any time.

Using the Health Center

Now that the JVM server is running, you can use the Health Center views to monitor the JVM server. The Health Center Environment perspective provides a convenient way to access all the Health Center function. It can be accessed by clicking **Windows** → **Open Perspective** → **Other** → **Health Center Environment**.

To connect to the Health Center agent that is running in the JVM, you must run the connection wizard. This wizard can be accessed from any of the Health Center perspectives by clicking **File** → **New Connection menu**.

The Health Center connection wizard starts and you see the window that is shown in Figure 8-22. Click **Next** and enter the information for the listening port (57108) as specified previously in the CICS JVM profile.

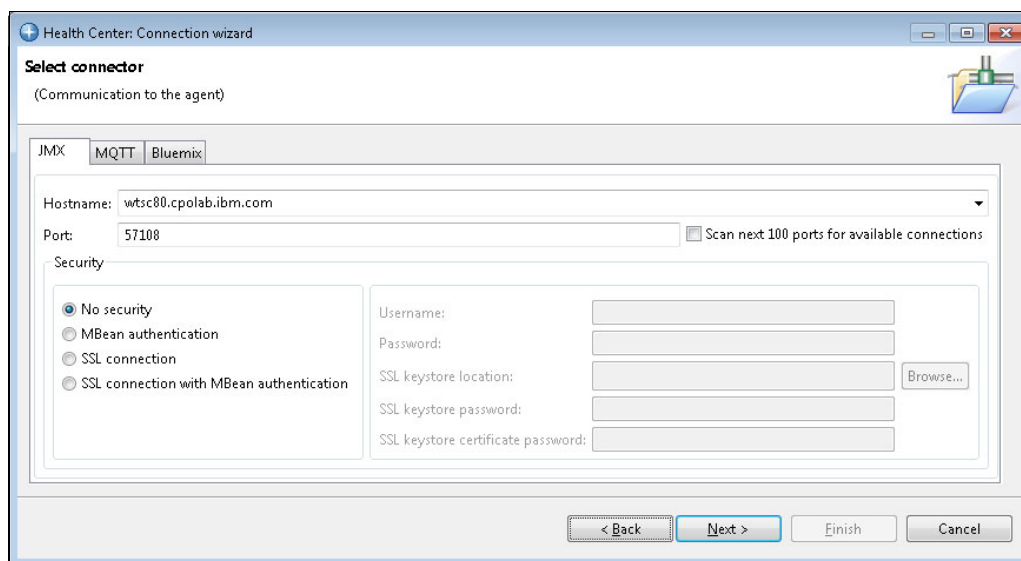


Figure 8-22 Health Center connection wizard

Click **Next** and the window in which you confirm the port that is being used by a Health Center agent on the host is shown (see Figure 8-23). Click **Finish**.

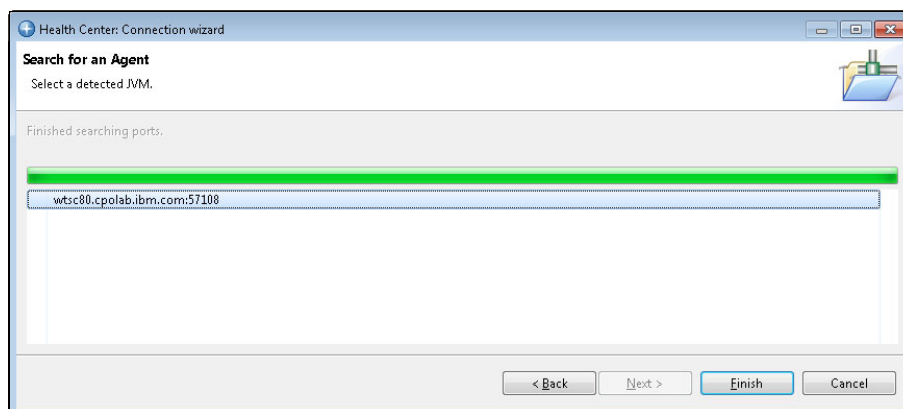


Figure 8-23 Connection confirmation

The main window of Health Center is shown in Figure 8-24.

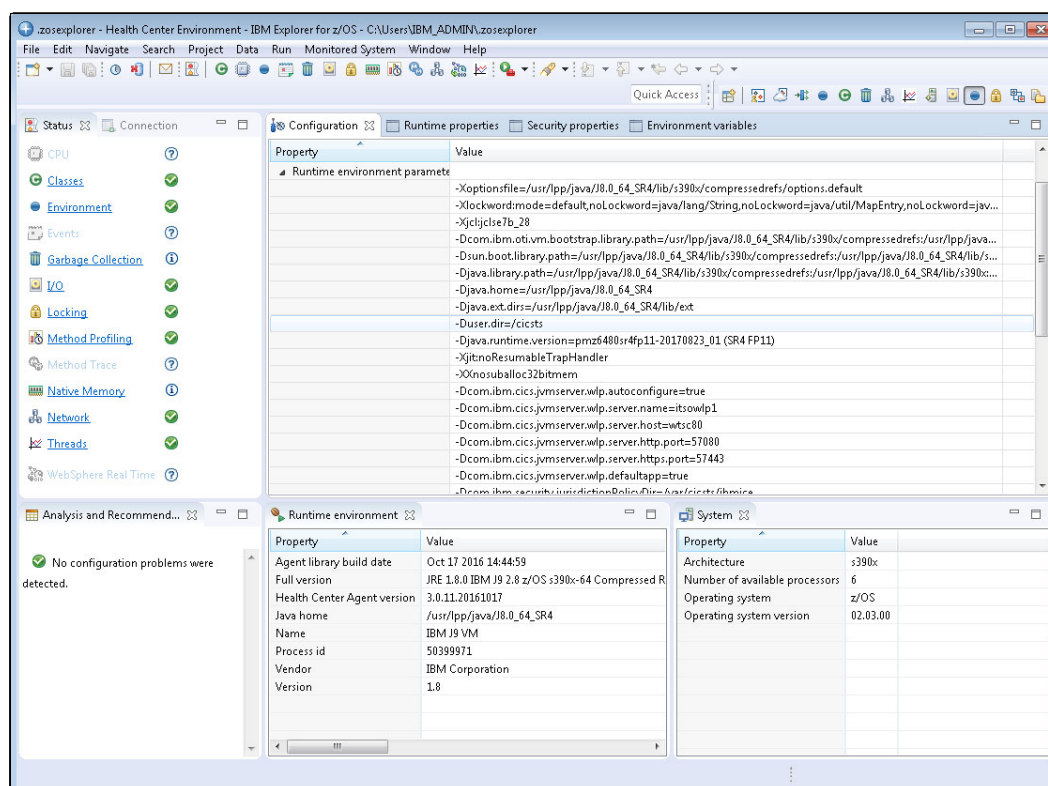


Figure 8-24 IBM Health Center main window

In the window that is shown in Figure 8-24, the general information regarding runtime environment, runtime properties, environment variables, security properties is shown. On the left side of the window, several links to information about the status of the following items are available:

- ▶ Environment information
- ▶ Classes
- ▶ Garbage collection
- ▶ I/O
- ▶ Locking
- ▶ Threads
- ▶ Native memory
- ▶ Method Profiling

Figure 8-25 on page 243 shows the Garbage Collection perspective. This perspective provides a set of views to assist in analyzing the collection (GC) process that is used by the JVM to manage memory in the JVM heap.

By using the default gencon GC policy the Java heap is split into two areas: the new or nursery area, and the old or tenured area. CICS JVM server statistics call new or nursery activity minor GC and call old or tenured activity major GC.

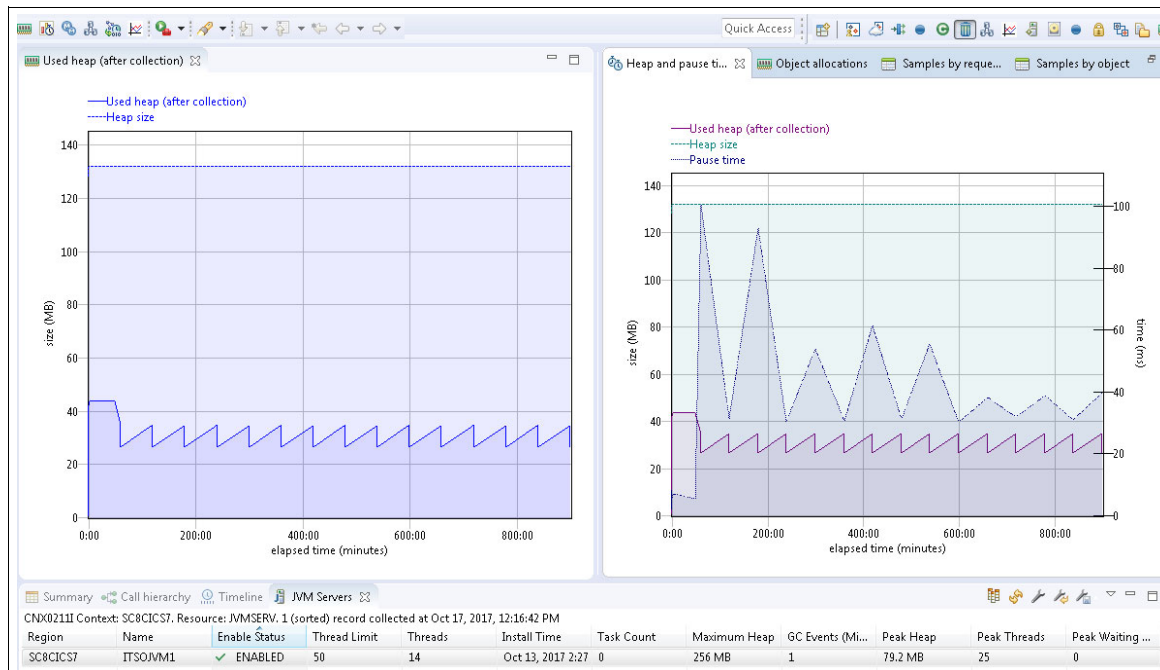


Figure 8-25 IBM Health Center garbage collection window

Much of this information is also displayed in the CICS Explorer JVM Servers view, which can be added to the Garbage Collection perspective by clicking **Window** → **Show view** → **Other** → **JVM Servers**. In addition, individual Health Center graphs for Used heap and Pause time can be added by using the same Show view menu. A customized perspective is created with a combination of Health Center and CICS views.

Custom perspectives such as these can be saved for future use by right-clicking the current perspective and selecting **Save As**, which adds the perspective to the list of available perspectives on the top row.

The Threads perspective provides detailed analysis of all the threads that are running within the JVM. A graphic view shows the thread count utilization against time.

The Current Threads view supplies a useful thread name filter, which allows the display to be filtered. For each thread that is displayed, the call stack can be displayed, which shows where each thread is suspended.

In our example that is shown in Figure 8-26 on page 244, we can see that CICS task 53525 is running under transaction ID CSMI and is suspended in a `Thread.sleep()` method that is named from the main method of the `JavaLinker()` class.

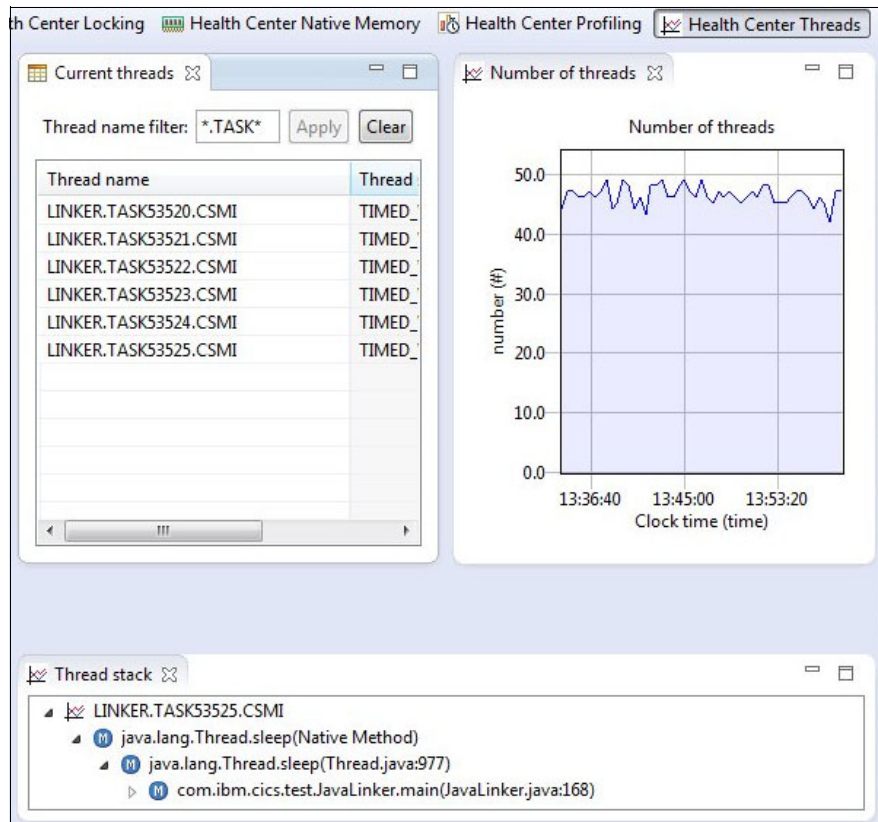


Figure 8-26 IBM Health Center threads window

8.2.6 IBM Application Metrics for Java

Application Metrics for Java (Javametrics) instruments the JVM in a Liberty server for performance monitoring, which provides data visually in a web-based dashboard.

Javametrics can be [downloaded from the GitHub website](#).

The following data collection sources are included:

- ▶ Environment: Machine and runtime environment information
- ▶ CPU: Process and system CPU
- ▶ GC: Percentage time that is spent in garbage collection
- ▶ Memory: Java native and non-native memory usage
- ▶ HTTP: HTTP request information

For more information, [see the GitHub repository](#).

To install Javametrics in our environment, we completed the following steps:

1. Downloaded the latest Javametrics release from GitHub, which includes the following components:
 - javametrics-dash-x.x.x.war (Javametrics Web Application)
 - javametrics-agent-x.x.x.jar (Javametrics agent)
 - ASM directory with related components
2. Copied the web application javametrics-dash-1.0.1.war file into our application deployment directory /var/cicsts/SC8CICS7/ITS0JVM1/apps.

3. Created a Javametrics directory in /var/cicsts/SC8CICS7/ITS0JVM1/app and copied the javametrics-agent-1.0.1.jar and asm folder into this directory (see Figure 8-27).

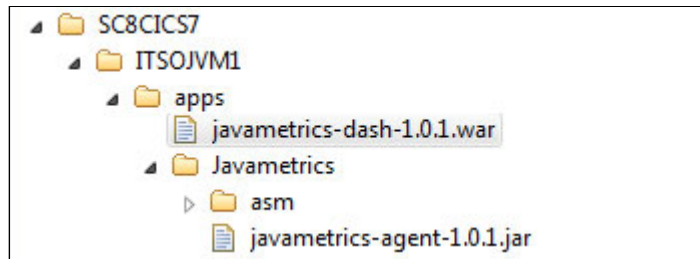


Figure 8-27 Javametrics application deployment directories

4. Added an <application> element to our server.xml file, as shown in Example 8-5.

Example 8-5 Javametrics deployed as an application in server.xml

```
<webApplication id="javametrics-dash-1.0.1"
  location="/var/cicsts/SC8CICS7/ITS0JVM1/apps/javametrics-dash-1.0.1.war"
  name="javametrics.dashboard">
</webApplication>
```

5. To start data collection, Javametrics requires a Java option to be set to load the agent JAR. In our scenario, the following line was added to the JVM profile for SC8CICS7 region:

```
-javaagent:/var/cicsts/SC8CICS7/ITS0JVM1/apps/Javametrics/
javametrics-agent-1.0.1.jar
```

After the application is installed, you should see the messages in the Liberty messages.log file that are shown in Example 8-6.

Example 8-6 Javametrics deployed as an application

```
CWWKZ0018I: Starting application javametrics-dash-1.0.1.
SRVE0169I: Loading Web Module: javametrics.web.
SRVE0250I: Web Module javametrics.web has been bound to default_host.
CWWKT0016I: Web application available (default_host):
http://wtsc80.cpolab.ibm.com:57080/javametrics-dash/
CWWKZ0001I: Application javametrics-dash-1.0.1 started in 0.307 seconds.
SESN0176I: A new session context will be created for application key
default_host/javametrics-dash
CWWKH0046I: Adding a WebSocket ServerEndpoint with the following URI:
/javametrics-socket
```

The URL for the dashboard consists of the server's default HTTP endpoint and javametrics-dash, as shown in the following example:

```
http://wtsc80.cpolab.ibm.com:57080/javametrics-dash/
```

Our Javametrics dashboard is shown in Figure 8-28 on page 246, in which you can see the following information:

- ▶ HTTP Incoming Requests: Response time measured in milliseconds.
- ▶ HTTP Throughput: Measured as requests per second (RPS).
- ▶ Top five average Response Times.
- ▶ Garbage Collection Time: Expressed in percentage over total along time.

- ▶ Heap size: Used memory, used native memory, total used memory, and used heap after GC along time.
- ▶ Environment information: Host name, number of processors, OS architecture.

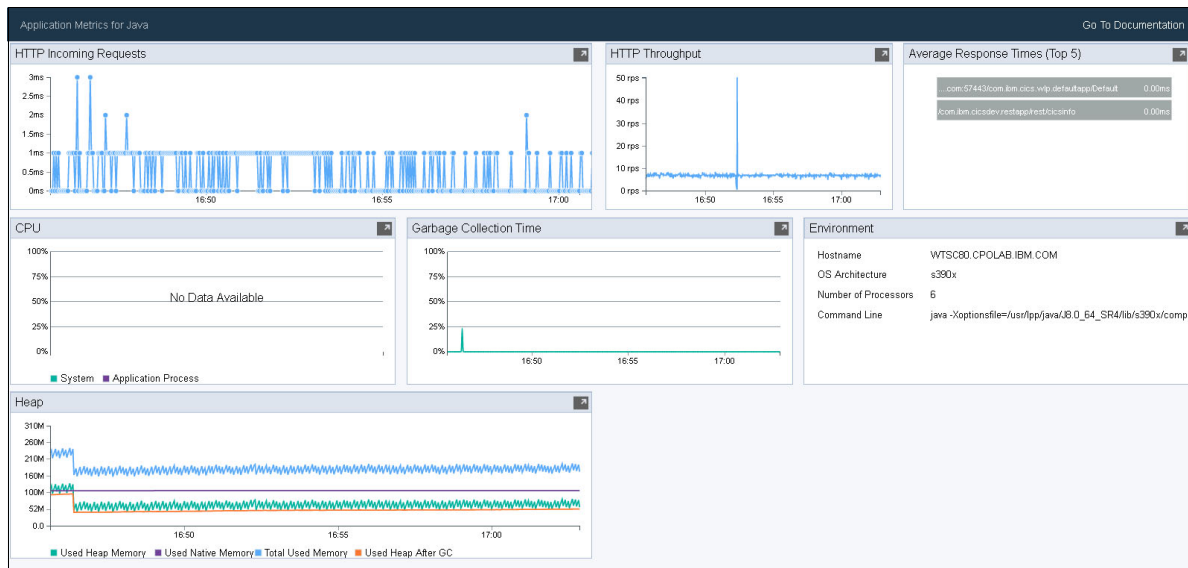


Figure 8-28 Javametrics Dashboard

Most of the data is plotted as the following line graphs:

- ▶ HTTP Incoming Requests, HTTP Outgoing Requests, and Other Requests show event duration against time.
- ▶ HTTP Throughput shows requests per second.
- ▶ Average Response Times shows the top five HTTP endpoints that on average take the longest to respond.
- ▶ CPU and Memory graphs show system and process usage over time.
- ▶ Heap shows the maximum heap size and used heap size over time.

A maximum of 15 minutes of data is shown across all graphs. If substantial data is being produced by the application that is monitored, the dashboard automatically aggregates the data.

If a graph includes points, hovering over one of these points provides more information. For example, HTTP Incoming Requests show the response time and the requested URL, as shown in Figure 8-29 on page 247.

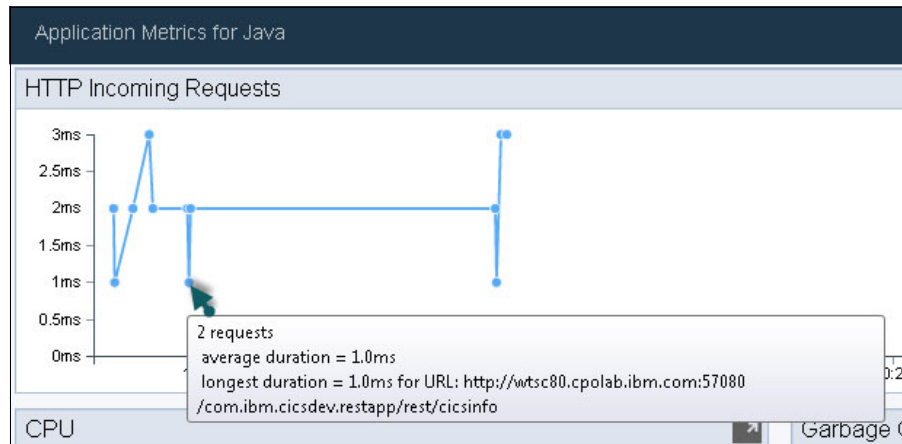


Figure 8-29 Hovering over a points to get more information

The Javametrics dashboard can help you to identify the following common performance problems:

- ▶ Slow HTTP response times on some or all routes
- ▶ Lower than expected throughput in the application
- ▶ Spikes in demand causing slowdown
- ▶ Higher than expected CPU usage for the level of throughput/load
- ▶ High or growing memory usage (potential memory leak)

8.2.7 Runaway tasks

To provide a system level mechanism to stop looping applications, the JVM server in CICS TS V5.4 now integrates CICS task runaway detection with the JVM server phase-out function.

Unlike traditional CICS tasks, a task that is running Java on a T8 TCB cannot be ended without consequences to other workload in the same JVM. Language Environment and the JVM server run in a POSIX-compliant environment, which mandates that if a thread is ended, the parent process is also ended. In turn, all child processes are ended abruptly, which causes all tasks in the JVM to fail immediately.

A task that is running in a JVM server that exceeds the RUNAWAY interval that is defined for the transaction experiences a more controlled stop process. This process differs from the traditional CICS behavior and you evaluate whether you want runaway intervals to apply to your Java tasks, or what value to set.

Note: This new runaway function is also included in CICS TS V5.3 with APAR PI77502.

JVM server runaway processing

When a task that is running Java experiences a runaway interval condition, CICS intercepts the condition and triggers a phase-out of the JVM server. New transactions are prevented from entering the JVM server and work is left to drain.

Then, if the task completes its processing, the JVM server enables again and becomes available for new requests. In many cases, if a task that is running in Java exceeds the runaway interval value, it is likely to be a bad application, such as a tightly looping application that prevents successful phase-out recycling of the JVM server.

When such a looping application is detected, the runaway timer triggers again after another interval and the JVM server phase out is escalated to a JVM server purge. Remaining tasks are subject to task purge processing and in most cases, are ended.

If further runaway intervals are exceeded, the JVM server purge escalates to a forcepurge and ultimately a kill until all running tasks are forcefully ended. The JVM server recycles back to the enabled state ready for new requests. If the JVM server must escalate as far as a kill request, it is prudent to recycle CICS at the earliest opportunity.

Tip: A runaway condition for a task that is running in a JVM server can cause temporary availability problems for the entire JVM server. For this reason, CICS modifies the runaway interval value that is specified for the transaction by multiplying it by a factor of 10 (up to a maximum value of 45 minutes). For example, if the TRANSACTION definition specifies RUNAWAY=5000, the effective runaway interval for that task when it runs in a JVM server is 50,000 milliseconds.

Setting the runaway interval value

By default, the CJSA transaction definition that is used for Liberty JVM servers has runaway detection active and set to the system interval. If you do not want runaway intervals to apply to these tasks, you can run work under your own transaction definitions set by using URIMAP definitions with the runaway interval set to 0, or another value of your choice.

8.2.8 CICS policies

In this section, we describe how a CICS policy can be defined and then deployed into a CICS region such that the policy rules are applied to all tasks in that region.

CICS TS V5.1 introduced the capability to define policies to monitor the resource utilization of a user task, and to automatically respond when resource usage exceeds the thresholds you define. In this way, excessive resource usage and certain types of looping and runaway transactions can be detected and managed.

A CICS policy is defined in a CICS bundle project and each bundle can define one or more policies. After the bundle is defined, it can then be deployed to a CICS region by using one of the following methods:

- ▶ At the region level.
- ▶ For a CICS Cloud application, at the platform level.
- ▶ For a CICS Cloud application at the application level.
- ▶ For a specific resource entry point, such as a URI map, transaction, or program to restrict all request that use that entry point.

We use the first method and deploy a policy at the region level to issue a message if a transaction runs for longer than 5 seconds. When this issue occurs, a message is written to the CICS log, which can then drive further automation, if required. Messages are sent to the CMPO destination, which by default is redirected to CSSL but can be directed to a separate log destination, if required.

Tip: When CICS policies are defined, we recommend the use of a minimum of CICS Explorer V5.4 because it supplies a new policy editor. This editor makes it able to view and edit policies after they are created.

Creating the policy

Complete the following steps to create a policy:

1. In CICS Explorer, open the Resource perspective. Click **File** → **New** → **CICS Bundle Project** and create a project with the name `elapsedtime.policy.bundle`, as shown in Figure 8-30.

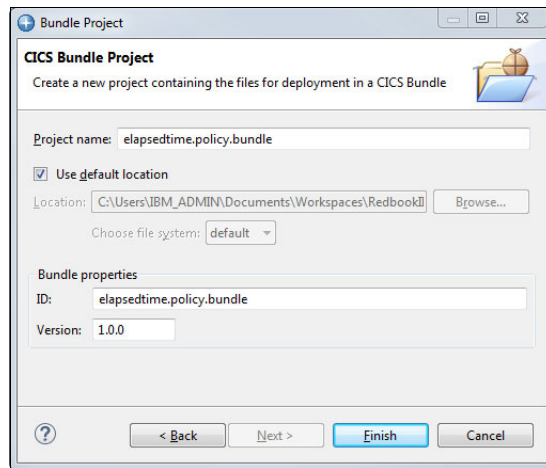


Figure 8-30 New CICS Bundle Project

2. Right-click the new bundle project and select **New** → **Policy Definition**.
3. In the Create Policy Definition window, enter `JVM_Elapsed.policy` as the file name and click **Finish** (see Figure 8-31).

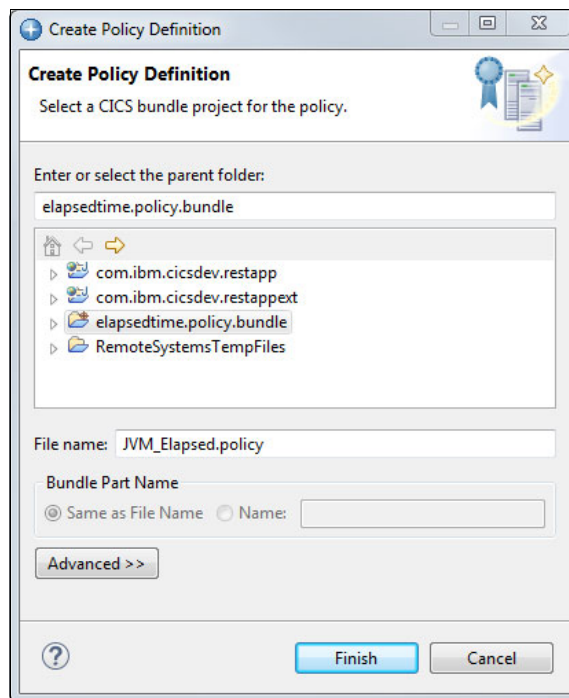


Figure 8-31 Creating a policy definition

4. When the Policy Overview window appears, enter the description JVM elapsed >5, as shown in Figure 8-32.

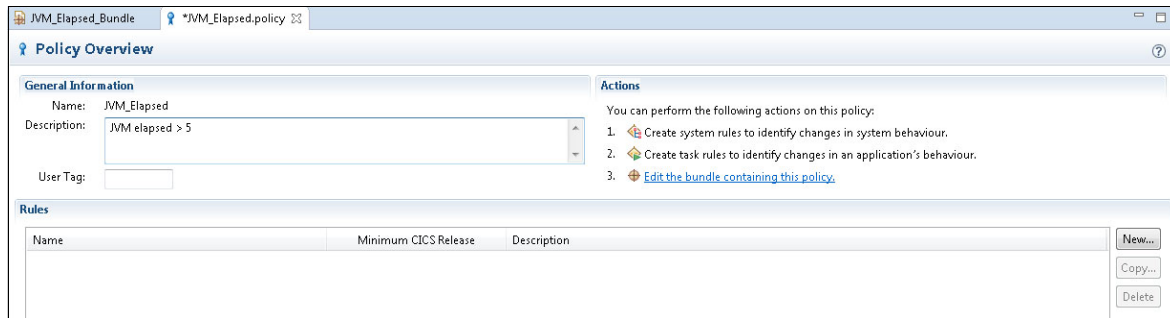


Figure 8-32 Policy definition tab

5. Click **New** to define a new rule. A pop-up window opens (see Figure 8-33) in which you are prompted to choose the type of rule. In our scenario, we chose **Task Rules** → **Time** to create a rule that is based on the elapsed time in a transaction (time rules can be based on elapsed time or CPU time).

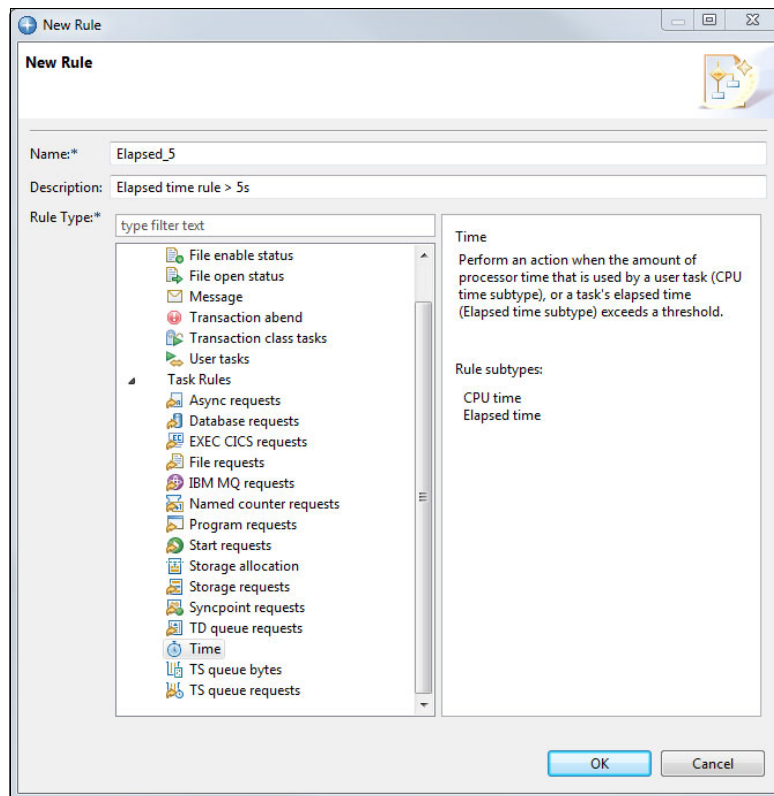


Figure 8-33 New policy rule

6. Click **OK**. The Rules editor opens in which you set the following conditions for the rule, as shown in Figure 8-34 on page 251.
 - a. Select **Elapsed time** in the condition drop-down list and enter 5 seconds for the greater than condition trigger.
 - b. Under Action, select the system default to **Issue a message**.
 - c. Close the editor and select **Yes** to save the rule.

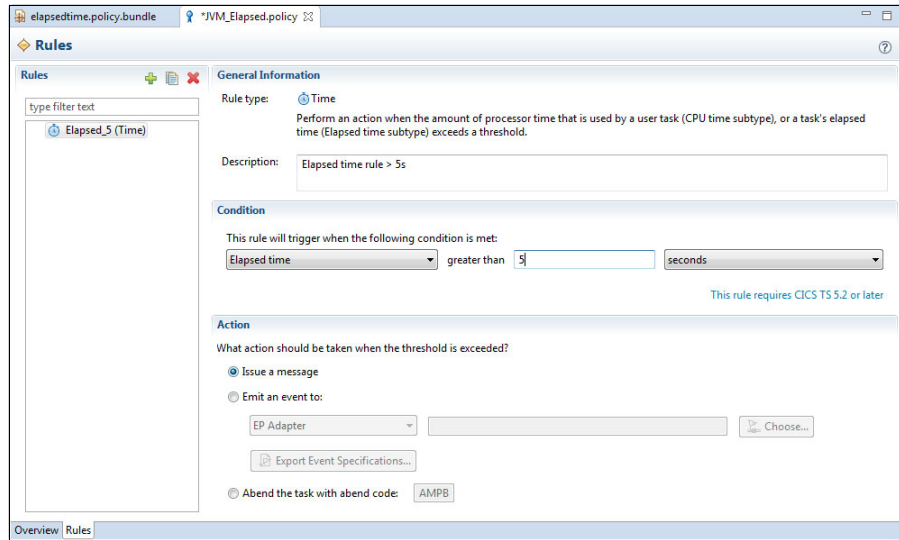


Figure 8-34 Defining policy rules

- Export the bundle project to a zFS location. We exported our bundle project `elapsedtime.policy.bundle` to the following CICS bundle deployment directory in zFS (see Figure 8-35):

`/var/cicsts/SC8CICS7/ITS0JVM1/bundles`

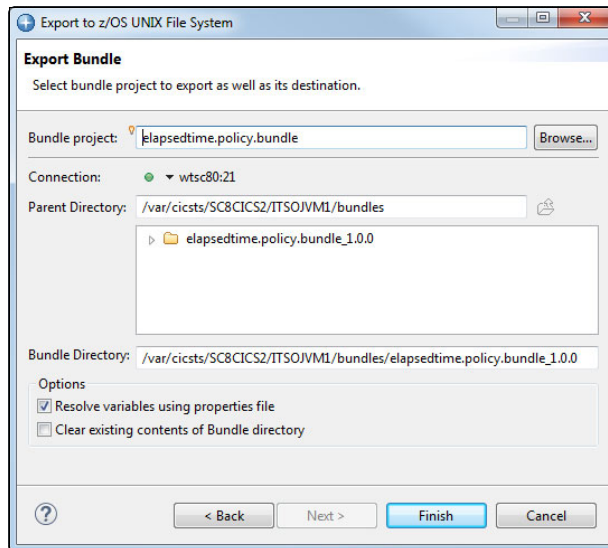


Figure 8-35 Export to zFS window

- This bundle must be installed into CICS. By using CEDA, create a CICS BUNDLE resource that is named JVMELA and set the bundledir attribute as the exported zFS directory elapsedtime.policy.bundle_1.0.0 in our JVM server bundle deployment directory, as shown in Figure 8-36.

```

OVERTYPE TO MODIFY                                CICS RELEASE = 0710
CEDA ALTER Bundle( JVMELA )
  Bundle      : JVMELA
  Group       : LIBERTY
  Description ==>
  Status      ==> Enabled          Enabled | Disabled
  Bundledir   ==> /var/cicsts/SC8CICS2/ITS0JVM1/bundles/elapsedtime.policy.b
  (Mixed Case) ==> undle_1.0.0
  ==>
  Basescope   ==>
  (Mixed Case) ==>
  ==>
  ==>
  DEFINITION SIGNATURE
  DEFInetime  : 11/29/17 15:11:33
+ CHANGETime  : 11/29/17 15:11:33
                                           SYSID=SC82 APPLID=SC8CICS2
PF 1 HELP 2 COM 3 END                6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL

```

Figure 8-36 CEDA define BUNDLE

- Install the BUNDLE JVMELA. It should enter the Enabled state.

If you enter the transaction CEMT INQ BUNDLE (see Figure 8-37), you see that the bundle JVMELA marked as Enabled and includes a Part count, Target count and Enabled count of 00001 indicates that the policy that is defined in the bundle part is active.

```

I BUNDLE(JVMELA)
STATUS: RESULTS - OVERTYPE TO MODIFY
  Bun(JVMELA ) Ena      Par(00001) Tar(00001)
    Enabledc(00001) Bundlei(elapsedtime.policy.bundle )

                                           SYSID=SC82 APPLID=SC8CICS2
RESPONSE: NORMAL                                TIME: 10.19.48 DATE: 11/30/17
PF 1 HELP 3 END                5 VAR          7 SBH 8 SFH 9 MSG 10 SB 11 SF

```

Figure 8-37 CEMT INQ BUNDLE

To test our policy, we ran a modified version of our restapp application taskInformation service. This application is modified to delay for 10 seconds within the Java code before running any JCICS commands. The JVM elapsed time policy then starts when the first JCICS command runs after the elapsed time for the transaction exceeds 5 seconds, as shown in Example 8-7.

Example 8-7 Message issued when policy triggered

```

DFHMP3001 11/30/2017 04:14:39 SC8CICS2 Task 04417(CJSA) exceeded a policy
threshold. BundleId=elapsedtime.policy.bundle, PolicyName=JVM_Elapsed,
RuleName=Elapsed_5, RuleType=time, Category=elapsedlimit, Threshold=5000000
(Value=5,Unit=S),
CurrentCount=10002168.

```

In this way, a policy can be used to monitor the activity of CICS transactions and use message automation to trigger other automated procedures.



Port sharing and cloning regions

In this chapter, we describe the considerations for sharing TCP/IP ports and other resources when cloned JVM servers are run in the same system. You learn how to configure TCP/IP port that are sharing to distribute HTTP connections.

You also learn how to share application configurations, SSL certificates, and Lightweight Third Party Authentication (LTPA) keys to make region cloning simpler.

This chapter includes the following topics:

- ▶ 9.1, “Sharing ports” on page 254
- ▶ 9.2, “Cloning regions” on page 258

9.1 Sharing ports

In an enterprise system environment, it is common for applications to be available on multiple host servers to distribute load efficiently or to make sure that core applications stay available during server outages (see Figure 9-1).

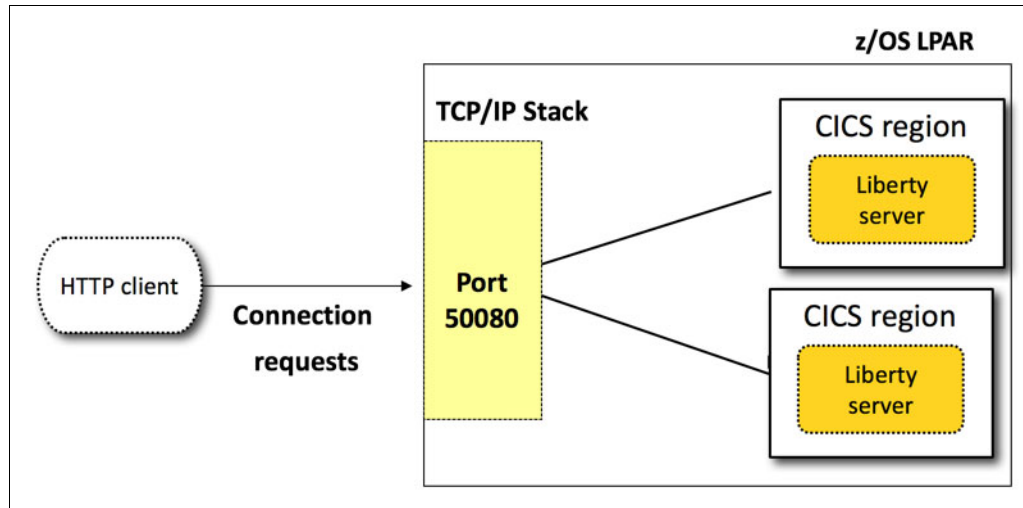


Figure 9-1 TCP/IP port sharing scenario

In this section, we describe how to set up TCP/IP port sharing for Liberty JVM servers, which allows incoming HTTP connections to the same application to be shared across multiple servers (assuming that the same application is installed on each server).

Note: TCP/IP port sharing requires that all listeners use the same TCP/IP stack; therefore, it is limited to Liberty JVM servers that are hosted in the same LPAR. If you want to share ports between Liberty JVM servers that are spread across multiple LPARs, you must use Sysplex Distributor.

First, you must decide on the port numbers you use for port sharing. For our examples, we use the following ports:

- ▶ 50080 for HTTP requests
- ▶ 50443 for HTTPS requests

We start by reserving the ports in our system's TCP/IP profile. In our system, we edited the following TCP/IP configuration profile data set:

SYS1.TCPPARMS (PROF80)

We specified 50080 and 50443 for port sharing. The two lines that are shown in Example 9-1 are the entries that we added to our TCP/IP profile.

Example 9-1 Port reservation statements in TCP/IP profile

```
50080 TCP SC8CICS* SHAREPORT ; CICS LIBERTY HTTP
50443 TCP SC8CICS* SHAREPORT ; CICS LIBERTY HTTPS
```

In Example 9-1, consider the use of the “*” suffix on the end of SC8CICS. In our system, all of our production CICS regions use the prefix SC8CICS. By using the “*”, any job that is prefixed with SC8CICS uses these ports.

After we save our changes, we must vary them active to ensure that they take effect. We entered the MVS command that is shown in Example 9-2 into SDSF on our host LPAR.

Example 9-2 TCP/IP vary command

```
/V TCPIP,,0,SYS1.TCPPARMS(PROF80)
```

Now that we varied these ports active, we can verify our changes by using the MVS command that is shown in Example 9-3. The example also shows the output of the command on our system.

Example 9-3 Output from our system verifying ports

```
/D TCPIP,,N,PORTL
```

```
RESPONSE=SC81
```

```
EZZ2500I NETSTAT CS V2R3 TCPIP 898
```

PORT#	PROT	USER	FLAGS	RANGE	IP ADDRESS	SAF NAME
-------	------	------	-------	-------	------------	----------

```
...
```

50080	TCP	SC8CICS*	DAS			
-------	-----	----------	-----	--	--	--

50443	TCP	SC8CICS*	DAS			
-------	-----	----------	-----	--	--	--

```
...
```

Now we must update our Liberty JVM servers in CICS to use these ports as HTTP endpoints, which can be done by using one of the following methods:

- If JVM server autoconfigure is used, we can update the system property `com.ibm.cics.jvmserver.wlp.server.http.port` in the JVM. Then, when the JVM server is restarted, it configures to use the ports as specified. How we set up this update in our JVM profile is shown in Example 9-4.

Example 9-4 Shared ports and autoconfigure settings in our JVM profile

```
-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true
```

```
...
```

```
-Dcom.ibm.cics.jvmserver.wlp.server.http.port=50080
```

```
-Dcom.ibm.cics.jvmserver.wlp.server.https.port=50443
```

- We can update our Liberty server configuration file to include a new HTTP endpoint. We do not need to change the endpoints that are used by the server. Instead, we add an HTTP endpoint for our shared ports. In our system, we used the XML elements in our servers that are shown in Example 9-5.

Example 9-5 XML configuration used to add in the additional shared ports in server.xml

```
<httpEndpoint id="sharingEndpoint"
  host="wtsc80"
  httpPort="50080"
  httpsPort="50443" />
```

When the server configuration is updated by Liberty again, you see a message in `messages.log` that indicates that the ports are now in use. The messages that our Liberty server produced are shown in Example 9-6.

Example 9-6 HTTP endpoint messages in messages.log

```
I CWWK00219I: TCP Channel sharingEndpoint has been started and is now listening
for requests on host wtsc80.cpolab.ibm.com (IPv4: 9.76.61.131) port 50080.
```

Now, when a request to an application comes into 50080 or 50443, it is routed to any Liberty JVM server with these ports defined.

Any application that you want to access in this manner is installed and available in all Liberty JVM servers. When a request is received, it is distributed between the active servers based on the following basic measurements that are provided by the server:

- ▶ The server's current connection count, which is the number of established TCP/IP connections.
- ▶ The server's backlog, which is the number of connection requests that are waiting to be accepted.
- ▶ SEF metric, which is the server efficiency accept fraction.

If security is active, you must ensure that your servers share the set of LTPA keys. For more information, see 9.2.4, “Sharing LTPA keys” on page 262. Without shared LTPA keys, you can encounter security errors that are caused by mismatching single sign-on tokens.

9.1.1 Using WLMHEALTH

Another option that is available to your systems for distributing incoming connections is to use z/OS workload manager (WLM) server-specific recommendations. Server-specific recommendations can be set by a CICS TS V5.4 region by using the WLMHEALTH SIT option, which helps the TCP/IP port sharing mechanism avoid CICS regions that are starting up or shutting down.

When WLMHEALTH is active in a CICS region, it gives any Liberty JVM servers time to start and install applications before receiving any work. Server-specific WLM uses the health value of the CICS regions to determine how to distribute work.

The health value starts at 0 and increments over time. CICS regions with a higher health value receive more work than those regions with a low health value thereby achieving a distribution of work that does not overburden any servers that are not yet ready.

After port sharing is set up, we can modify the TCPIP profile for the LPAR to use WLM. Find the entries you added to `SYS1.TCPPARMS(PROF80)`. These entries resemble the entries that are shown in Example 9-7.

Example 9-7 Entries in the TCPIP profile

```
50080 TCP SC8CICS* SHAREPORT ; CICS LIBERTY HTTP
50443 TCP SC8CICS* SHAREPORT ; CICS LIBERTY HTTPS
```

We modify these entries to use WLM. For each port that is used, edit the entries to change SHAREPORT to SHAREPORTWLM. The resulting entries are shown in Example 9-8.

Example 9-8 Updated entries in the TCPIP profile for WLM

```
50080 TCP SC8CICS* SHAREPORTWLM ; CICS LIBERTY HTTP
50443 TCP SC8CICS* SHAREPORTWLM ; CICS LIBERTY HTTPS
```

As in the previous scenario, you must vary on these entries to activate the changes. This activation is done by running the MVS command that is shown in Example 9-9.

Example 9-9 Command vary the TCPIP profile

```
/V TCPIP,,0,SYS1.TCPPARMS(PROF80)
```

Now that our ports are set up to use WLMHEALTH, we update the CICS job to include a SIT override parameter that specifies some basic thresholds for WLM.

WLM can use server-specific recommendations to estimate the health of any specific server. During the startup process, CICS increments the health value of the server at specific intervals. This health rating then helps TCP/IP determine where to send new connection requests.

The WLMHEALTH SIT parameter is used to set the increment period and the rate at which the health measurement is increased. The value we used for WLMHEALTH is shown in Example 9-10.

Example 9-10 WLMHEALTH SIT parameter in our region's JCL

```
WLMHEALTH= (30,25)
```

The first value in this SIT parameter is the time interval after startup at which the WLM health is changed. In our example, we set the value to 30, which means that CICS increases the health from zero 30 seconds after start. The second value is the percentage change for each time interval.

The result in our case is that the health of our CICS regions increases every 30 seconds by a value of 25. The health value of our CICS region during the start process is listed in Table 9-1.

Table 9-1 Changes in WLM health over time

Seconds after start	Health value
0	0
30	25
60	50
120	100

As HTTP connections arrive at our shared ports, they are distributed to the Liberty servers based on the health value of each CICS region. CICS regions that are starting do not receive any TCP/IP connection requests when the health is 0 if other regions are listening on the shared port with a health greater than 0.

In our configuration, this configuration allows our Liberty JVM servers a period of 30s after start to fully initialize and install the required applications. It also includes another 90 seconds until they are operating at maximum capacity.

During start, the z/OS WLM health rating is printed to the console log for each region. How a set of sample messages from our CICS region SC8CICS7 is shown in Example 9-11.

Example 9-11 Example messages for WLM health

```
DFHMN0115I SC8CICS7 CICS Server z/OS WLM Health percentage is now 25.  
DFHMN0115I SC8CICS7 CICS Server z/OS WLM Health percentage is now 50.  
DFHMN0115I SC8CICS7 CICS Server z/OS WLM Health percentage is now 75.  
DFHMN0115I SC8CICS7 CICS Server z/OS WLM Health percentage is now 100.
```

9.2 Cloning regions

When cloning CICS regions, the administration overhead of creating and managing the systems is easier if the configured resources are also shared. When Liberty JVM servers are used, the following key configuration files can be shared:

- ▶ CSD file
- ▶ JVM profile
- ▶ Liberty server.xml

CSD file

A CICS region CSD file (DFHCSD) can be shared between cloned regions. The same JVMSERVER resource definition can be used to install a cloned JVM server into multiple regions.

JVM profile

The JVM profile in zFS can be shared between CICS region. It is located by using the JVMProfile attribute in the JVMSERVER definition, which is then used to locate the file from the zFS directory that is named in the JVMPROFILEDIR SIT option.

Note: JVM profiles can be shared only between servers if no unique values are included in the profile. Examples of such unique values are TCP/IP ports that cannot be shared, such as those values used by debuggers or IBM Health Center.

Liberty server.xml

The CICS region's Liberty JVM server's configuration file (server.xml) cannot be shared because CICS must read and write to this file. However, most of the configuration it contains can be shared.

The main mechanism for sharing parts of a server's configuration file is by using the `<include>` XML element. You can create separate XML files that contain distinct pieces of configuration that can then be added or removed from a server's configuration, as required. In the following sections, we describe some of the configuration parts that you might share, and provide you with guidelines for making the process it easier.

9.2.1 Sharing application definitions

You might find that many of your Liberty JVM servers share a set of applications. By using `<application>` elements in server.xml with the `<include>` element, you can create lists of these applications that can then be easily shared between servers.

It is recommended that you keep any shared applications in the same directory in HFS. We used the following location for storing all of our shared applications:

```
/var/cicsts/SC8CICS/wlp/shared/apps
```

We also recommend keeping shared configuration files, such as the application list we are about to create, in a shared directory similar to the directory that was created for our shared applications. In our system, the following location was used:

```
/var/cicsts/SC8CICS/wlp/shared/config
```

Next, we create an XML file that contains a list of applications, which are defined by using the application element. The application element is a piece of configuration that specifies an application to be installed in Liberty.

We use the file name `sharedApps.xml` for our file and we add two application elements (and thus two applications) to the list. The contents of our `sharedApps.xml` file can be seen in Example 9-12.

Example 9-12 Contents of sharedApps.xml

```
<server>

<application
location="/var/cicsts/SC8CICS/wlp/shared/apps/com.ibm.cicsdev.restapp.war"/>

<application
location="/var/cicsts/SC8CICS/wlp/shared/apps/com.ibm.cicsdev.restappext.war"/>

</server>
```

The server elements surround the application elements. The server elements are required for Liberty to incorporate the configuration. To add this piece of configuration to a Liberty JVM server in CICS, we add an `<include>` element to its `server.xml` file, as shown in Example 9-13.

Example 9-13 The <include> element in server.xml

```
<server>

<include location=
"/var/cicsts/SC8CICS/wlp/shared/config/sharedApps.xml"/>

</server>
```

When the Liberty JVM server next updates its configuration, it includes `sharedApps.xml` in its processing. You see a message in `messages.log` that informs you it included this file in its configuration update. The message that we received from our Liberty JVM server is shown in Example 9-14.

Example 9-14 Example include message in messages.log

```
Processing included configuration resource:
/var/cicsts/SC8CICS/wlp/shared/config/sharedApps.xml
```

9.2.2 Sharing SSL configuration

It is common that Liberty JVM servers spread across multiple CICS regions must use the same keystore for validating incoming certificates and storing their server certificates.

These keystores can be shared with multiple Liberty JVM servers individually configured for the same keystore. Alternatively, you can create a sharable configuration file for your SSL settings. By using this configuration, you can share the keystores by adding an include element to the `server.xml` file of your servers.

SSL settings are configured through the following sets of XML elements in `server.xml`:

- ▶ The `<ssl>` configuration element, which specifies the parameters of the SSL settings for that server.
- ▶ One or more `<keystore>` elements, which point the server at stores of certificates or user credentials.

These two sets of elements can easily be separated out into a separate XML file, which can then be referenced by using an include element in your `server.xml` file. In Example 9-15, you can see an example file that is named `sslConfig.xml`. This file contains SSL settings and a RACF keystore for our servers to use.

Example 9-15 XML for the `sslConfig.xml` file

```
<ssl clientAuthentication="false" id="defaultSSLConfig"
      keyStoreRef="defaultKeyStore" sslProtocol="TLSv1.2"/>

<keystore fileBased="false" id="defaultKeyStore"
          location="safkeyringhw://CICSREGN/LIBERTY.SC8CICS7"
          password="password" readOnly="true"
          type="JCECCARACFKS"/>
```

If you plan to share RACF key rings, it is important to note that the CICS region user ID must be the same for each of the regions that share the key ring. If the IDs are different, you might encounter RACF access errors.

With a separated SSL configuration file created, we can now include it in our server configurations. We can do so by using the `<include>` element in our `server.xml` files. Set the location attribute to the absolute path of the configuration file, as shown in Example 9-16.

Example 9-16 `<include>` element in `server.xml` that points to the `datasources.xml` configuration file

```
<include location=
"/var/cicsts/SC8CICS/wlp/shared/resources/sslConfig.xml"/>
```

When the server configuration is next refreshed, the changes take effect. You see a message informing you that the configuration was updated. All of the HTTPS endpoints must be restarted by Liberty as part of this change, which is done automatically. You see messages informing you of the restart in the `messages.log` file. The messages for the configuration update and the HTTPs endpoint restart from our system are shown in Example 9-17 on page 261.

Example 9-17 Messages in messages.log indicating the HTTPS endpoints restarted

```
A CWWKG0017I: The server configuration was successfully updated in 0.076 seconds.
I CWWK00220I: TCP Channel defaultHttpEndpoint-ssl has stopped listening for
requests on host wtsc80.cpolab.ibm.com (IPv4: 9.76.61.131) port 57443.
I CWWK00220I: TCP Channel sharingEndpoint-ssl has stopped listening for requests
on host wtsc80.cpolab.ibm.com (IPv4: 9.76.61.131) port 50443.
I CWWK00219I: TCP Channel defaultHttpEndpoint-ssl has been started and is now
listening for requests on host wtsc80.cpolab.ibm.com (IPv4: 9.76.61.131) port
57443.
I CWWK00219I: TCP Channel sharingEndpoint-ssl has been started and is now
listening for requests on host wtsc80.cpolab.ibm.com (IPv4: 9.76.61.131) port
50443.
```

9.2.3 Sharing feature configuration

The features list in a Liberty JVM server often is similar between servers. This similarity occurs because all of your servers rely on a set of core technologies in the system to run their applications. As a part of configuration sharing, you can choose to share feature lists or individual features.

The procedure is straightforward. Start by deciding on the list of features that you want to share. This list can be a full list of features on its own, or you want to share the related feature with the resource. For example, you have a shared data source so it might include the JDBC-4.1 feature in the same file.

We create a `features.xml` file in our zFS, which contains a set of features that we want to include in the configuration of every server in our system. The contents of this file is shown in Example 9-18.

Example 9-18 XML configuration for features.xml

```
<featureManager>
  <feature>jdbc-4.1</feature>
  <feature>jaxrs-2.0</feature>
  <feature>jsonp-1.0</feature>
</featureManager>
```

In our system, we keep this file in a shared directory to which all Liberty JVM servers have read access. This location in our file system is `/var/cicsts/SC8CICS/wlp/shared/config`.

To use this file as part of a server's configuration, we add an `include` element to the server's `server.xml` file. The `include` element that we used in our environment is shown in Example 9-19.

Example 9-19 Include statement in server.xml which adds our features.xml to the configuration

```
<include location=
"/var/cicsts/SC8CICS/wlp/shared/config/features.xml"/>
```

When the server configuration is refreshed, you see a message in `messages.log` that indicates the features that were installed in your JVM server. An example of this message is shown in Example 9-20.

Example 9-20 Example message for feature enablement in `messages.log`

```
A CWWKF0012I: The server installed the following features: [servlet-3.1,
beanValidation-1.1, ssl-1.0, jndi-1.0,appSecurity-2.0, jdbc-4.1, jaxrs-2.0,
cicsts:link-1.0, cicsts:security-1.0, jsonp-1.0, cicsts:core-1.0, json-1.0,
wab-1.0, websocket-1.1].
I CWWKF0008I: Feature update completed in 22.741 seconds
```

The list of newly installed features includes the features `jdbc-4.1`, `jsonp-1.0`, and `jaxrs-2.0`.

You do not need to be concerned about duplicated features because Liberty recognizes which features are in use. However, you must ensure that feature lists do not mix Java EE 6 and Java EE 7 features because it can lead to conflicts.

9.2.4 Sharing LTPA keys

LTPA is a type of authentication token that is supported by Liberty servers. When a user successfully authenticates with a Liberty server that supports LTPA, a token is generated by the server that confirms that the specific user ID successfully passed the authentication step. That token is then returned to the caller as a cookie, which can be sent back to the server on the next call or to another server, as shown in Figure 9-2.

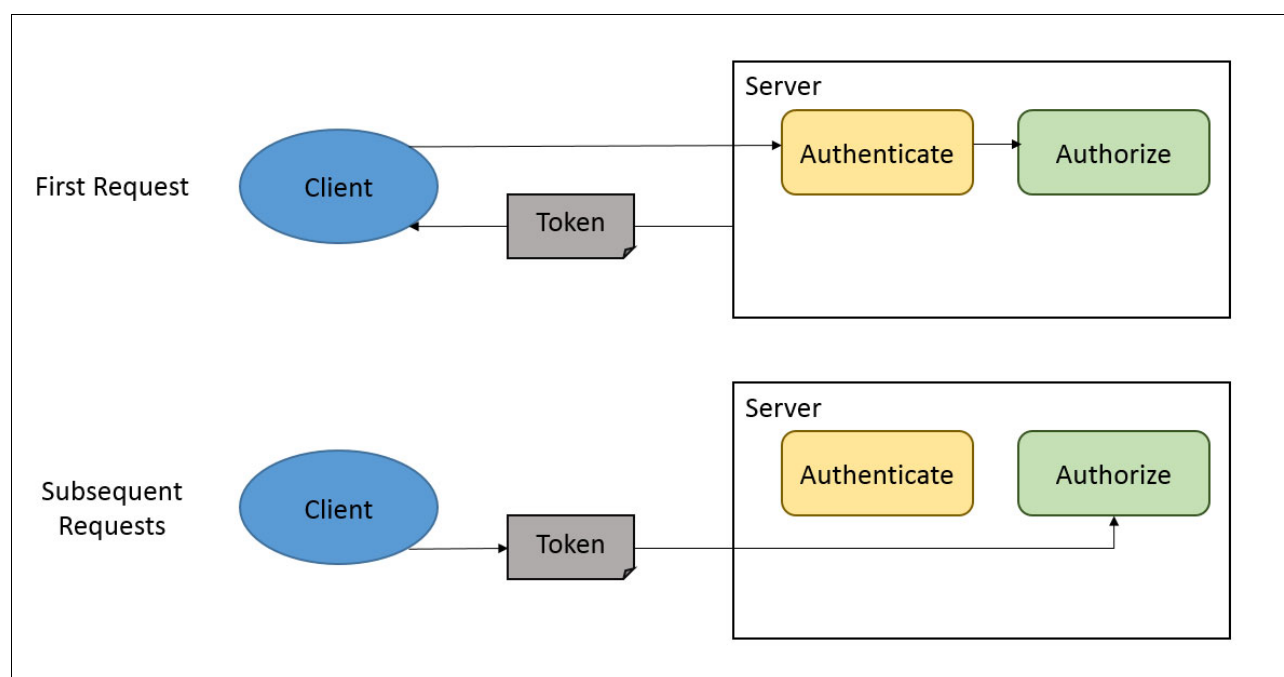


Figure 9-2 Basic security flow with an LTPA token

When a Liberty server receives an authentication token, it verifies the validity of the token by checking it against some private keys, which are often stored in the server's file structure. If the validity is confirmed, the authentication step is skipped and the user ID moves to the authorization step.

By sharing LTPA keys across servers, you can maintain the identity that is associated with a session, even if the user's request is directed to a different server because of workload balancing or other causes.

If you do not share the LTPA keys and leave each server to generate and manage their own keys, you can encounter errors in the authentication process. LTPA tokens that are generated in one server are invalid in another server.

The key component of making this mechanism work is to ensure that all servers that use the LTPA tokens agree on a shared set of keys. This agreement is ensured by using one of the following methods:

- ▶ Copy the LTPA keys from one file system to another and configure each server to use their own set of duplicated keys.
- ▶ Store the LTPA keys in a shared file location and configure each server to use the same key file.

In this scenario, we focus on the second option because it better fits the scenarios that are described in this IBM Redbooks publication. It is also arguably more secure than the first option because the keys are never transmitted over a network. This configuration minimizes the chance of unauthorized access.

First, we need a location for storing the LTPA keys that all Liberty JVM servers can access. In our environment, we use the following shared directory to share resources across all of our Liberty JVM servers:

```
/var/cicsts/SC8CICS/wlp/shared
```

LTPA keys are considered a security resource in Liberty, and so are usually stored in the corresponding shared directory. In our system the following directory is used:

```
/var/cicsts/SC8CICS/wlp/shared/resources/security
```

After this directory is created, your `server.xml` file's LTPA configuration must be updated to point at keys in this directory. You do not need to create the keys. If a set of keys that match your configuration are not available, Liberty generates them for you. This process occurs at start or after a configuration refresh if the server is running.

In our server, the LTPA configuration is shown in Example 9-21.

Example 9-21 Simple LTPA configuration with shared keys in server.xml

```
<ltpa
  keyFileName="/var/cicsts/SC8CICS/wlp/shared/resources/security/ltpa.keys"
  keyPassword="{xor}Lz4sLCgwLTs="/>
```

Every server that you want to use LTPA tokens must be configured to use the same set of keys in the same directory. Liberty handles the lifecycle of these keys as and when it is needed. When keys expire, a new set is generated. Because all servers share the keys file, all servers immediately are updated.

Redbooks

Liberty in IBM CICS: Deploying and Managing Java EE

SG24-8418-00

ISBN 073844216X



(0.5" spine)

0.475" <-> 0.873"

250 <-> 459 pages



SG24-8418-00

ISBN 073844216X

Printed in U.S.A.

Get connected

