

Implementing IBM CICS JSON Web Services for Mobile Applications

Includes architectural patterns and
example scenarios

Illustrates integration with IBM
Worklight

Is based on CICS TS Feature
Pack for Mobile Extensions



Rufus Credle
Andy Armstrong
Chris Atkinson
Russell Bonner
Geoff Pirie
Inderpal Singh
Nigel Williams
Matthew Wilson
Mark Woolley



International Technical Support Organization

Implementing IBM CICS JSON Web Services for Mobile Applications

November 2013

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

First Edition (November 2013)

This edition applies to IBM CICS Transaction Server® V5.1 and IBM Worklight® Server 6.0.

© Copyright International Business Machines Corporation 2013. 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!	xii
Comments welcome	xiii
Stay connected to IBM Redbooks	xiii
Part 1. Introduction and architecture	1
Chapter 1. Introduction	3
1.1 Overview	4
1.2 Business value	4
1.3 Solution overview	5
1.4 Solution architecture	7
1.5 Usage scenarios	8
1.6 Integration of CICS and other IBM products or solutions	9
1.7 Supported platforms	9
Chapter 2. CICS use of mobile technologies	11
2.1 REST	12
2.2 JSON	12
2.3 Existing support in CICS for mobile	12
2.3.1 Atom	13
2.3.2 SOAP web services	13
2.4 New mobile support in CICS	13
2.4.1 JSON with feature pack	13
Chapter 3. CICS and IBM Worklight	15
3.1 Overview	16
3.2 Introduction to IBM Worklight	16
3.2.1 The Worklight platform	17
3.2.2 Further reading	19
3.3 Using Worklight with CICS	19
3.3.1 Architecture overview	19
Chapter 4. Patterns for JSON in CICS	21
4.1 CICS as a JSON web service provider	22
4.1.1 Starting with an existing application (bottom-up)	22
4.1.2 Starting with an existing JSON interface (top-down)	23
4.2 CICS as a client for JSON web services	25
4.2.1 Integrating other JSON web services into your CICS application	26
4.2.2 How CICS supports acting as a client for JSON web services	26
4.3 Handling JSON in other CICS applications	27
Part 2. Setup and configuration	29
Chapter 5. Configuring CICS for the example scenarios	31
5.1 Comparison with SOAP web services	32

5.2 CICS as a service provider	32
5.2.1 How CICS processes a request	32
5.2.2 How to configure CICS as a service provider	34
5.3 CICS as a JSON client	46
Chapter 6. IBM Worklight configuration	47
6.1 Worklight Studio	48
6.2 Worklight Server	52
Chapter 7. Security and workload management	69
7.1 Security overview	70
7.1.1 Security principals and concepts.	70
7.1.2 CICS security options for JSON web services	71
7.1.3 CICS mobile security topologies	73
7.1.4 Worklight security	74
7.1.5 DataPower security.	75
7.1.6 IBM Endpoint Manager for Mobile Devices.	76
7.2 Configuring security for JSON web services.	76
7.2.1 Configuring the URIMAP.	76
7.2.2 Configuring the TCPIPSERVICE	77
7.3 Worklight security configuration	78
7.3.1 Configuring HTTP basic authentication in Worklight.	78
7.3.2 Configuring SSL in Worklight	80
7.4 Workload management overview	80
7.5 Workload balancing.	81
7.6 TCP/IP load balancing techniques	82
7.6.1 Port sharing.	82
7.6.2 Virtual IP addressing.	82
7.6.3 Sysplex Distributor	82
7.7 JSON web services and business logic: A multi-region approach	83
Chapter 8. Problem determination.	85
8.1 Introduction	86
8.2 Deployment problems	86
8.3 Problems with the JSON assistants	87
8.3.1 DFHJS2LS	87
8.3.2 DFHLS2JS	88
8.4 Problems with requests to JSON web services.	88
Part 3. Application development and scenarios	91
Chapter 9. Language structure to JSON schema scenario	93
9.1 General insurance sample application	94
9.2 Use case for language structure to JSON.	94
9.3 Language Structure to JSON schema solution.	94
9.3.1 Identifying the COBOL programs and copybooks.	95
9.3.2 Tailoring DFHLS2JS for the COBOL customer programs.	96
9.3.3 Submitting the DFHLS2JS JCL.	98
9.3.4 Enabling the JSON Request URI	100
9.3.5 Test that the JSON request can be successfully performed.	101
Chapter 10. JSON schema to language structure scenarios	105
10.1 JSON web services: Request-Response and RESTful.	106
10.2 JSON web services: A use case	106

10.3 Request-Response JSON web service implementation	107
10.3.1 Reviewing the JSON schema	107
10.3.2 Mapping the JSON schema to language structures	109
10.3.3 Submitting the DFHJS2LS JCL	111
10.3.4 Developing the CICS wrapper application program	112
10.3.5 Defining the CICS resources.	112
10.3.6 Testing the application	114
10.4 RESTful JSON web service implementation.	115
10.4.1 Reviewing the JSON schema	116
10.4.2 Mapping the JSON schema to language structures	117
10.4.3 Submitting the DFHJS2LS JCL	119
10.4.4 Developing the CICS wrapper application program	120
10.4.5 Defining the CICS resources.	120
10.4.6 Testing the application	122
Chapter 11. Developing a simple JSON web service client application	127
11.1 Overview of the solution	128
11.1.1 The scenario	128
11.1.2 The solution	128
11.1.3 The linkable interface for transforming JSON	129
11.2 Writing the JSON schema.	130
11.2.1 Writing the request schema	132
11.2.2 Writing the response schema	134
11.3 Generating the language structures	135
11.4 Defining the CICS resources.	137
11.4.1 Defining the BUNDLE resources.	137
11.4.2 Defining the URIMAP resource.	140
11.5 Developing the application program	142
11.5.1 Transforming the request data	143
11.5.2 Sending the request	145
11.5.3 Transforming the response body	146
11.6 Testing the sample application	148
Chapter 12. IBM Worklight for CICS	149
12.1 Creating a Worklight adapter	150
12.2 Testing the Worklight adapter	154
12.3 Calling the Worklight adapter from the Worklight client code	156
Part 4. Appendix	161
Appendix A. Sample level for a JSON schema.	163
Sample JSON schema generated from COBOL customer create program.	164
Appendix B. Sample COBOL programs	167
Sample programs for CICS as a client for JSON web services	168
Sample client application	168
Sample provider application	173
Appendix C. Additional material	175
Locating the web material	175
Using the web material.	175
Downloading and extracting the web material	175

Related publications	177
IBM Redbooks publications	177
How to get IBM Redbooks publications	177
Help from IBM	177

Notices

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

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

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

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

BladeCenter®	DB2®	Redpapers™
CICS®	developerWorks®	Redbooks (logo)  ®
CICS Explorer®	IBM®	System x®
CICSplex®	Parallel Sysplex®	System z®
Cognos®	RACF®	Tealeaf®
Coremetrics®	Rational®	WebSphere®
DataPower®	Redbooks®	z/OS®

The following terms are trademarks of other companies:

Worklight is trademark or registered trademark of Worklight, an IBM Company.

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

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

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

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

Preface

This IBM® Redbooks® publication provides information about how you can connect mobile devices to IBM Customer Information Control System (CICS®) Transaction Server (CICS TS), using existing enterprise services already hosted on CICS, or to develop new services supporting new lines of business. This book describes the steps to develop, configure, and deploy a mobile application that connects either directly to CICS TS, or to CICS via IBM Worklight® Server. It also describes the advantages that your organization can realize by using Worklight Server with CICS.

In addition, this Redbooks publication provides a broad understanding of the new CICS architecture that enables you to make new and existing mainframe applications available as web services using JavaScript Object Notation (JSON), and provides support for the transformation between JSON and application data. While doing so, we provide information about each resource definition, and its role when CICS handles or makes a request.

We also describe how to move your CICS applications, and business, into the mobile space, and how to prepare your CICS environment for the following scenarios:

- ▶ Taking an existing CICS application and exposing it as a JSON web service
- ▶ Creating a new CICS application, based on a JSON schema
- ▶ Using CICS as a JSON client

This Redbooks publication provides information about the installation and configuration steps for both Worklight Studio and Worklight Server. Worklight Studio is the Eclipse interface that a developer uses to implement a Worklight native or hybrid mobile application, and can be installed into an Eclipse instance. Worklight Server is where components developed for the server side (written in Worklight Studio), such as adapters and custom server-side authentication logic, run.

CICS applications and their associated data constitute some of the most valuable assets owned by an enterprise. Therefore, the protection of these assets is an essential part of any CICS mobile project. This Redbooks publication, after a review of the main mobile security challenges, outlines the options for securing CICS JSON web services, and reviews how products, such as Worklight and IBM DataPower®, can help. It then shows examples of security configurations in CICS and Worklight.

Authors

This book was produced by a team of specialists working at IBM Hursley Park, Hursley, England.



Rufus Credle is a Certified Consulting IT Specialist at the International Technical Support Organization (ITSO), Raleigh Center. In his role as project leader, he conducts residencies and develops IBM Redbooks publications and IBM Redpapers™ publications.

He has expertise in subjects including network operating systems, enterprise resource planning (ERP) solutions, voice technology, high availability, clustering solutions, web application servers, pervasive computing, IBM and original equipment manufacturer (OEM) e-business applications, IBM WebSphere® Commerce, IBM industry technology, IBM System x®, and IBM BladeCenter®.

Rufus' various positions during his IBM career include assignments in administration and asset management, systems engineering, sales and marketing, and IT services. He has a Bachelor of Science degree in Business Management from Saint Augustine's College. Rufus has been employed at IBM for 33 years.

Follow Rufus on Twitter: <http://twitter.com/rcredle1906>

Join Rufus's network on LinkedIn:
<http://www.linkedin.com/pub/rufus-p-credle-jr/1/b/926/>



Andy Armstrong is a CICS System Test - Senior Inventor with IBM Software Group, Application and Integration Middleware Software in IBM Hursley Park, United Kingdom. He is responsible for testing CICS in the way a client would use the product. He develops new testing tools, integrates new tests, and help others shape their tests.

Andy has experience with regression testing and automated testing, code coverage tools, and Eclipse development skills. He also has IBM System z® skills.



Chris Atkinson has worked for IBM for over 13 years as a Software Engineer on IBM z/OS® middleware (CICS and IBM WebSphere MQ). He has over 10 years of software engineering experience in testing CICS Transaction Server and associated products. Previously, Chris was a CICS application and support programmer for a financial company.



Russell Bonner is an IBM Certified Consulting IT Specialist. In his current role as a Client Technical Specialist with the IBM Software Group based in the UK, he is a technical consultant to clients and IBM Business Partners. He specializes in the System z portfolio of software solutions, with a focus on the Application and Integration Middleware Software family. He is also a member of the European CICS Software Architecture team.

Russell has presented at numerous events and conferences, including Guide Share Europe UK and Innovate UK, regarding WebSphere, IBM Rational®, and CICS Transaction Server technologies, and associated application development and integration solutions.



Geoff Pirie is a CICS product marketing manager with the IBM Software Group - Application and Integration Middleware Software located at IBM Hursley Park, United Kingdom. He joined IBM as a developer and later moved into the management of the CICS System Test and Performance team.

Today, his experience and growth led him to join the marketing team for CICS, enjoying and experiencing the management of the CICS portfolio.



Inderpal Singh joined the CICS Transaction Server development team in 2009. He has worked as a software engineer in many areas, including CICS System Management, cloud enablement, and IBM CICS Explorer®. His most recent role was as the Team Lead for the CICS Transaction Server Feature Pack for Mobile Extensions.

After studying at The University of Sheffield, Inderpal received a Bachelor of Science degree in Computer Science and Artificial Intelligence.



Nigel Williams is a Certified IT Specialist working in the New Technology Center, Montpellier, France. He specializes in security, CICS, and enterprise application integration. He is the author of many papers and Redbooks publications, and speaks regularly on security and CICS topics.



Matthew Wilson joined the CICS development team in 2008, and is based in the Hursley lab in the UK. He has worked in a wide range of areas, including storage management, 64-bit application support, and the CICS Explorer. Most recently, he is the lead developer for the CICS Transaction Server Feature Pack for Mobile Extensions.

He holds a Master of Computer Science with Distributed Systems and Networks from Southampton University.



Mark Woolley is a member of the IBM CICS Level 3 Service team at the IBM Hursley Laboratory in the UK. In addition to working on the delivery of the Mobile Extensions Feature Pack, Mark has focused on resolving problems with WebSphere MQ support in CICS. Before joining IBM, Mark gained a Bachelor of Science degree in Computer Science from the University of Bath in the UK.

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

Tamikia Barrow-Lee, Richard Conway, Robert Haimowitz
ITSO, Raleigh and Poughkeepsie Center

Mark Cocker, CICS Technical Strategy and Planning, IBM Software Group, Application and Integration Middleware Software
IBM Hursley

Jenny He, Software Engineer, CICS Development, IBM Software Group, Application and Integration Middleware Software
IBM Hursley

Daniel Fitzgerald, former Software Engineer, CICS Development, IBM Software Group, Application and Integration Middleware Software
IBM Hursley

Stewart Smith, CICS Systems Test, IBM Software Group, Application and Integration Middleware Software
IBM Hursley

Paul Cooper, Software Developer, IBM Software Group, Application and Integration Middleware Software
IBM Hursley

Regis David, Client Technical Specialist, CICS & WebSphere MQ Technical Solution Architect, IBM Sales & Distribution, Software Sales
IBM France

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.

Obtain more information about the residency program, browse the residency index, and apply online at the following website:

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 the following website:

ibm.com/redbooks

- ▶ Send your comments in an email:

redbooks@us.ibm.com

- ▶ Mail your comments:

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

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

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

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

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

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

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

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

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



Part 1

Introduction and architecture

This part introduces and provides information about the CICS mobile strategy, and the IBM Worklight Server technology.



Introduction

Our goal with this IBM Redbooks publication is to provide all of the information necessary for you to connect mobile devices to IBM Customer Information Control System Transaction Server (CICS TS). You can do this whether you want to maximize existing enterprise services already hosted on CICS, or to develop new services supporting new lines of business.

This book describes the steps necessary to develop, configure, and deploy a mobile application that connects either directly to CICS Transaction Server, or to CICS using IBM Worklight Server. In addition, we will show you how you can use CICS to operate as a requester of JavaScript Object Notation (JSON)-based services.

With this book, you will be able to understand the key architectural decisions associated with making CICS services available to mobile devices, and find example code to quickly get up and running. Our main scenarios are based on the general insurance application (GENAPP) Support Pack (CB12), which you can download and try for yourself, following the examples in this book. Download it from the following website:

<http://www-01.ibm.com/support/docview.wss?uid=swg24031760>

The following topics are covered in this chapter:

- ▶ 1.1, “Overview” on page 4
- ▶ 1.2, “Business value” on page 4
- ▶ 1.3, “Solution overview” on page 5
- ▶ 1.4, “Solution architecture” on page 7
- ▶ 1.5, “Usage scenarios” on page 8
- ▶ 1.6, “Integration of CICS and other IBM products or solutions” on page 9
- ▶ 1.7, “Supported platforms” on page 9

1.1 Overview

For many years CICS Transaction Server has been capable of hosting mobile enterprise services. The introduction of web services capabilities in CICS Transaction Server V3 provided the fundamental building blocks of service connectivity, enabling the adoption of service-oriented architecture (SOA), and underpinning today's mobile solutions.

CICS has continued to add new capabilities to the run time. From a mobile perspective, the introduction of the CICS TS Feature Pack for Mobile Extensions provides JSON and Representational State Transfer (REST)-conforming (RESTful) web service support, further enhancing the options for enterprise applications to mobile devices.

Customers around the world use CICS TS to host hundreds of millions, and in some cases billions, of transactions per day. As the number of mobile devices worldwide continues to grow, so does the variety and volume of workload that they drive. CICS has the capacity to scale up in support of this increasing mobile workload, providing an exceptional platform for hosting mobile workloads.

1.2 Business value

By extending existing enterprise applications onto a mobile platform, your business can capitalize on its existing investment without the need to develop an entirely new solution to support mobile services. In addition, a line of business can now offer service to users who increasingly expect to be able to interact with a company using their mobile phone.

As a platform, the primary benefits offered by CICS in support of mobile devices are noted in the following list:

- ▶ Provide reuse of existing enterprise services.
- ▶ Using the established web service technology within CICS, it is relatively simple to build a set of enterprise services that can be used by a mobile device.
- ▶ Provide simplified consumption of enterprise data using JSON-formatted data.

A common misconception is that enterprise data in CICS can be hard to use. The CICS TS Feature Pack for Mobile Extensions provides support for JSON data, which is rapidly becoming the standard format for data interchange on mobile devices.

- ▶ CICS already operates at the heart of the enterprise.

Hosting mobile applications within CICS brings them closer to the enterprise data that they are accessing, minimizing application path lengths and keeping response times down.

- ▶ Adopt a RESTful architectural style for service delivery.

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

- ▶ Provide capacity to manage mobile workload.

We noted earlier that customers around the world use CICS TS to host hundreds of millions, and in some cases billions, of transactions per day. CICS Workload Management provides a robust and scalable platform suitable for supporting the heaviest of mobile workloads.

1.3 Solution overview

This book introduces four different approaches to building mobile services in CICS TS. In two of these three cases, details are provided for how to connect the mobile application, either directly to CICS or via IBM Worklight Server. Chapter 6, “IBM Worklight configuration” on page 47 describes in more detail the architectural implications of connecting the mobile application, either directly or indirectly, via IBM Worklight Server.

The different approaches that we describe in detail are summarized in the following list:

- ▶ The *top-down* approach. Figure 1-1 shows the suggested method of building new enterprise services for a mobile application in CICS. This approach lends itself to the RESTful architectural style. This approach enables you to create a set of services with a concise interface. For more information about the RESTful architectural style, see the following website:

<http://www.ibm.com/developerworks/library/wa-aj-multitier/>

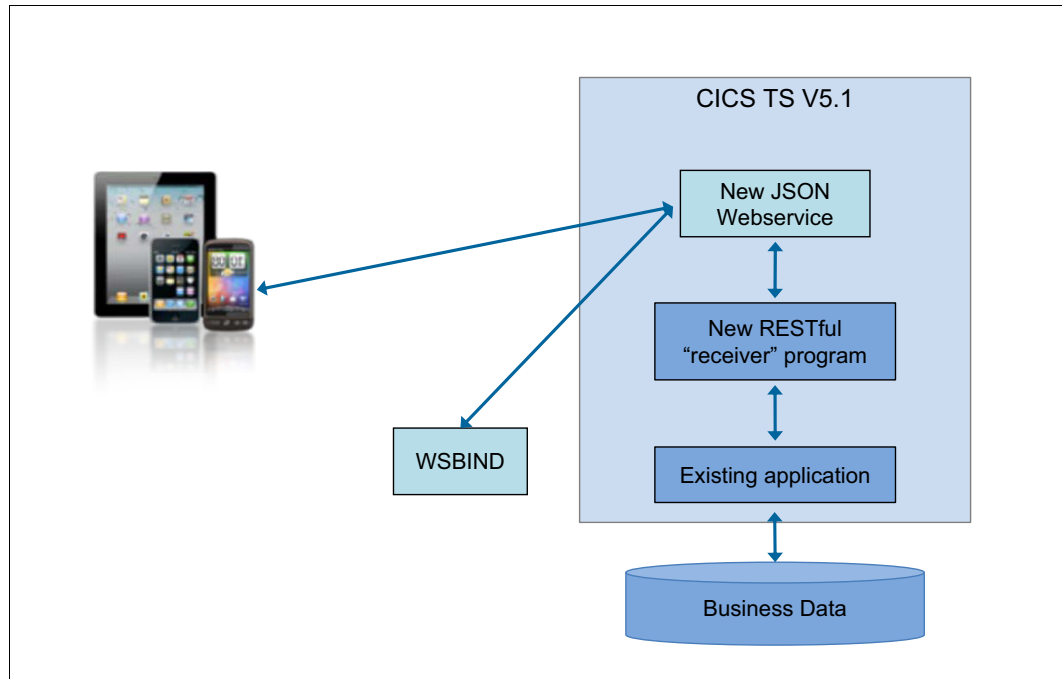


Figure 1-1 A possible way to implement a JSON web service starting from the JSON schema

- ▶ The *bottom-up* approach. Figure 1-2 on page 6 shows perhaps the fastest approach for delivering enterprise services to mobile devices. Building on an existing SOA, the bottom-up approach enables you to define a JSON or SOAP interface to an existing Common Business Oriented Language (COBOL), C/C++, PL/I, or Java application. This approach maximizes the reuse of existing assets, and minimizes the creation of new components.

Figure 1-2 shows the bottom-up approach.

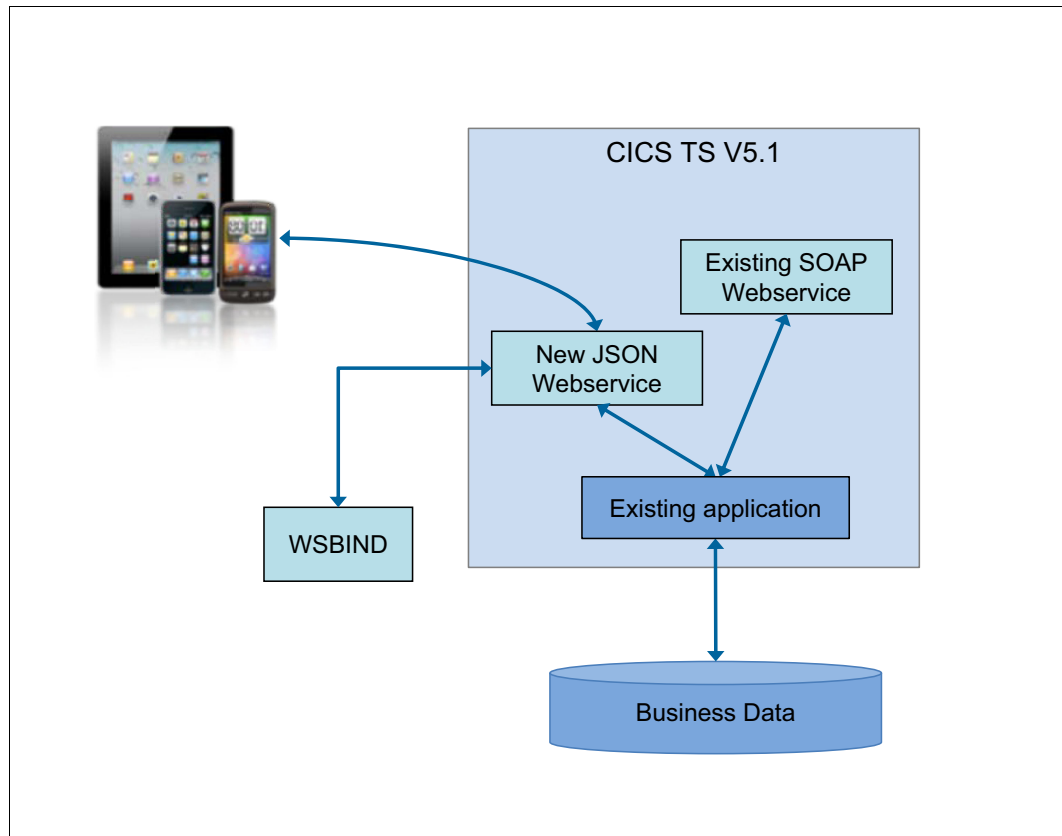


Figure 1-2 Architecture for a JSON web service to be driven by high-level language data structure

- ▶ The *requester mode* approach. Figure 1-3 shows how this approach enables CICS to participate in JSON-based interactions, and to make requests against external service providers that offer a JSON-based interface. Though not strictly a pure mobile scenario, the capabilities offered by the CICS TS Feature Pack for Mobile Extensions provide CICS with additional options for connecting to the wider enterprise using JSON-formatted data.

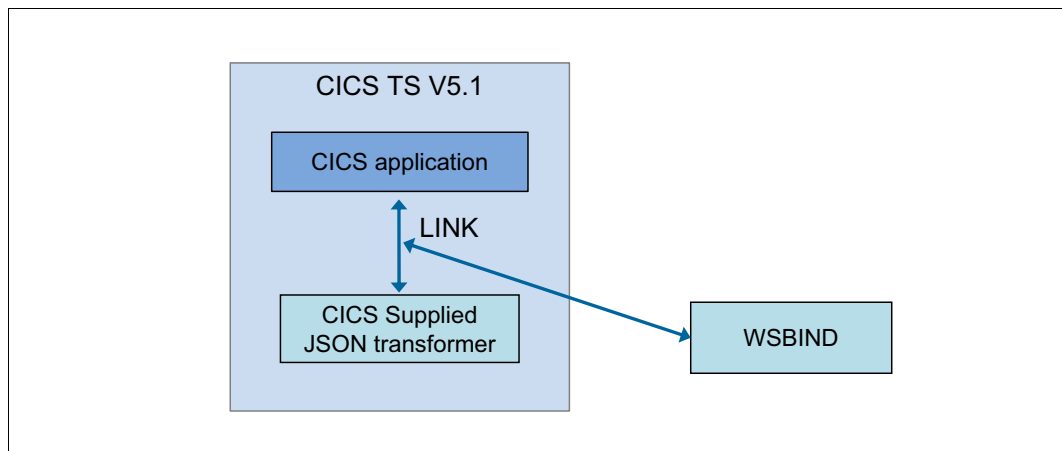


Figure 1-3 LINKable CICS program transforms data between high-level data structures and JSON

- ▶ Java API for RESTful Web Services (JAX-RS) is a programming interface that provides support in creating web services according to the REST architectural pattern, as shown in Figure 1-4. REST is an architecture style for designing networked applications without the need for complex mechanisms, such as Common Object Request Broker Architecture (CORBA) or SOAP.

The pattern involves client/server communications where the state of an application is held by the client, which reduces processing required on the server. Using the IBM WebSphere Liberty profile provided in CICS TS V5.1, you are able to write your business applications using JAX-RS.

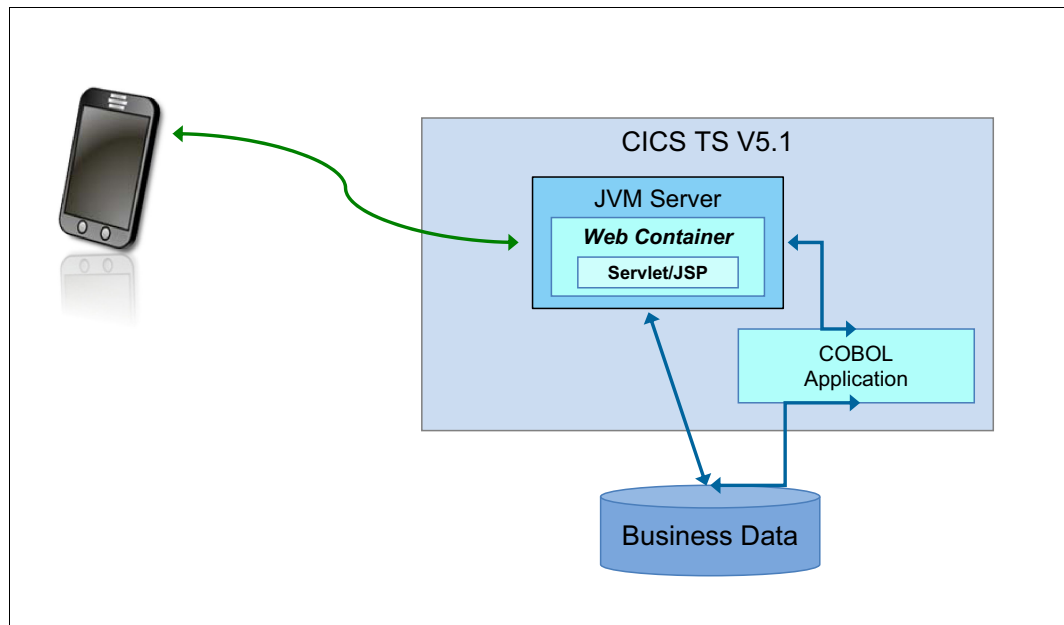


Figure 1-4 JAX-RS provides a rapid and easy development of RESTful-enabled Java applications

1.4 Solution architecture

The architecture for an enterprise mobile solution based on CICS will vary, depending on business requirements and the business data that the applications require. This book provides information about and demonstrates the following solution architectures:

- ▶ Direct to CICS
- ▶ A two-tier Worklight and CICS solution

The direct-to-CICS solution is one where the mobile devices communicate directly to CICS. In this architecture, other devices communicate with CICS through the existing web services provided by CICS. This scenario is ideal when the devices and networks involved are all trusted, the applications involved do not require frequent updates, and the applications do not run on multiple platforms.

An alternative solution is an architecture with one or more layers between CICS and the user devices. Worklight is a solution that provides governance and security for your mobile applications, along with a powerful software development kit (SDK) for rapid development of your enterprise applications on most major platforms.

In this architecture, the mobile devices communicate with the Worklight Server, which ensures that the device has access rights to make requests to CICS. If approved, this request is then sent to CICS, and the CICS application is run.

IBM Worklight also manages the versioning of applications, enabling new versions of the application to be created without the need for multiple versions of the back-end business applications, each with logic to handle the different requests. On the different platforms, features, such as notifications, are also handled and standardized by Worklight.

Figure 1-5 shows a typical architecture of how Worklight and CICS TS can be used in conjunction to extend the reach of your CICS applications to a mobile platform.

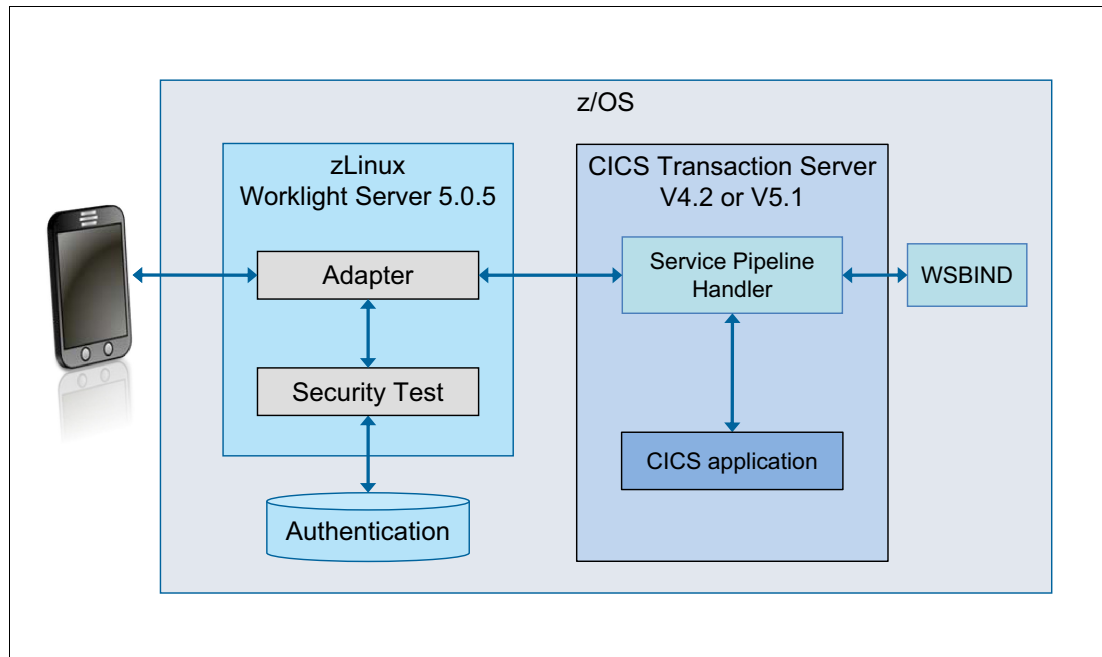


Figure 1-5 Mobile devices access services hosted on CICS TS using Worklight and CICS TS

1.5 Usage scenarios

The following scenarios show several ways in which CICS TS can be used to solve enterprise mobile business solutions:

- ▶ As an insurance company, you identify a *requirement to enable your policy holders to view and make claims on their policies directly from their mobile device*. To remain competitive, the mobile application needs to be available as soon as possible. To facilitate rapid development, and to make your services hosted on CICS available to the application, you choose the bottom-up approach.

Using your COBOL copybooks, you generate a JSON schema that enables the mobile application to communicate with CICS web services with a lightweight payload. CICS manages the transformation between JSON and the COBOL copybook structure, and your CICS services have been made available with ease.

- ▶ As a CICS Service provider, you have been informed of a business *requirement to make your CICS applications available through a standardized RESTful pattern*. By using the top-down approach, you externalize your existing and new CICS services through a RESTful architecture. This enables your services to be called through a unified approach understood throughout the business.

- ▶ As an airline carrier, you have a *requirement to access data from your partner companies to accurately allocate seating through your on-demand ticket purchasing system*. You are informed that your partner company only externalizes their services through a RESTful pattern, with JSON as the data format of choice.

You use the new CICS-provided LINKable program, along with the existing CICS WEB API, to communicate with the partner company. These tools enable you to communicate without the added cost of development, while maintaining a bespoke communication layer.

1.6 Integration of CICS and other IBM products or solutions

CICS enterprise mobile solutions can be deployed in a product stack with the following IBM products:

- ▶ Worklight is a hybrid mobile solution offering governance, and a powerful SDK to build applications with a server component that will drive the future mobile world.
- ▶ IBM DataPower enables you to secure, integrate, and optimize SOA capabilities that scale.

CICS enterprise mobile solutions also function with the following existing solutions that work with CICS:

- ▶ WebSphere MQ
- ▶ IBM DB2®
- ▶ IBM Integration Bus (formerly WebSphere Message Broker)

1.7 Supported platforms

CICS web service support is available from CICS TS V3 and later. For further details about CICS TS V3 requirements, see the following website:

<http://www-01.ibm.com/software/htp/cics/tserver/sysreqs/>

CICS TS Feature Pack for Mobile Extensions V1.0 is available on CICS TS V4.2 and CICS TS V5.1. For further details of the requirements of CICS TS Feature Pack for Mobile Extensions V1.0, see the following website:

<http://publib.boulder.ibm.com/infocenter/prodguid/v1r0/clarity-reports/report/html/softwareReqsForProduct?deliverableId=1358442733620&osPlatform=z/OS>



CICS use of mobile technologies

This chapter describes the existing and new aspects of Customer Information Control System Transaction Server (CICS TS) that enable you to move your CICS applications, and business, into the mobile space. The chapter includes the following topics:

- ▶ 2.1, “REST” on page 12
- ▶ 2.2, “JSON” on page 12
- ▶ 2.3, “Existing support in CICS for mobile” on page 12
- ▶ 2.4, “New mobile support in CICS” on page 13

2.1 REST

Representational State Transfer (REST) is a defined set of architectural principles by which you can design web services that focus on service resources. The REST architectural pattern takes advantage of the technologies and protocols of the World Wide Web to describe how data objects can be defined and modified.

In contrast to a request-response model such as SOAP, which focuses on procedures made available by the system, REST is modeled around the resources in the system. Each resource is globally identifiable through its Uniform Resource Identifier (URI). Because REST does not focus on the procedures and services provided by a system, a small number of actions are defined based on the existing Hypertext Transfer Protocol (HTTP) methods: GET, POST, PUT, DELETE, HEAD. The methods are used as shown in the following list:

GET	Retrieve a resource representation.
PUT	Modify a resource representation.
POST	Create a new resource representation.
DELETE	Delete a resource representation.
HEAD	Retrieve a resource's metadata.

It is important to notice that REST does not carry any information regarding a service in the HTTP Body of a request.

2.2 JSON

JavaScript Object Notation (JSON) is an open standard format for data interchange. Although originally used in the JavaScript scripting language JSON is now language-independent, with many parsers available in many languages.

Compared to Extensible Markup Language (XML), JSON has many advantages. Most predominantly, JSON is more suited to data interchange. XML is an extremely verbose language: Every element in the tree has a name, and the element must be enclosed in a matching pair of tags.

Alternatively, JSON expresses trees in a nested array format similar to JavaScript. This enables the same data to be transferred in a far smaller data package with JSON than with XML. This lightweight data package lends itself to better performance when parsing.

JSON supports two structures: Objects and arrays. Objects are an unordered collection of name-value pairs, where arrays are ordered sequences of values. JSON also supports four simple types: Strings, numbers, Boolean expressions, and null values. This enables JSON to describe any resource. JSON can be seen as both human and machine-readable. JSON is an easy language for humans to read, and for machines to parse.

2.3 Existing support in CICS for mobile

CICS has been providing web services capabilities since CICS TS V3. The first capability introduced was SOAP web services in CICS TS V3.1. Atom support followed in CICS V4.1.

2.3.1 Atom

The Atom Syndication Format and the Atom Publishing Protocol are two standards that together make the Atom standard. CICS can provide Atom feeds using data provided by CICS resources. Atom feeds supply web clients with a series of data items containing metadata for each item in the Atom Syndication Format. With CICS V4.1, your CICS applications can be enabled to provide live information for Web 2.0 consumption. HTTP requests can also be used to edit CICS resources following the Atom Publishing Protocol.

Atom enables you to access your CICS resources in a REST-conforming (RESTful) way using XML without a heavyweight process.

For information about Atom feeds from CICS, see the following website:

http://pic.dhe.ibm.com/infocenter/cicsts/v4r1/index.jsp?topic=%2Fcom.ibm.cics.ts.ineternet.doc%2Ftopics%2Fdfhtl_atom_serving.html

2.3.2 SOAP web services

SOAP is a simple XML-based protocol for applications to exchange information over Application Layer protocols such as HTTP. SOAP can be used to create request-response interactions. SOAP is a lightweight protocol which is platform, operation system, and transport-independent.

SOAP web services support was introduced in CICS TS V3.1

This XML-based protocol consists of the following three parts:

- ▶ An envelope, which defines what is in the message and how to process it
- ▶ A set of encoding rules for expressing instances of application-defined data types
- ▶ A convention for representing procedure calls and responses

The most common method of exchanging SOAP messages uses HTTP. However, SOAP can be used with a variety of transport protocols, such as Java Message Service (JMS), Simple Mail Transfer Protocol (SMTP), or File Transfer Protocol (FTP).

2.4 New mobile support in CICS

CICS TS for IBM z/OS Feature Pack for Mobile Extensions V1.0 introduced new capabilities to CICS web services.

2.4.1 JSON with feature pack

Support has now been introduced, through the feature pack, to enable CICS to accept and receive HTTP web service requests using the JSON data format when calling your CICS applications using CICS web services. CICS can be configured so that your existing CICS applications can take advantage of this without any need for the application to be updated. This configuration can be driven from either a JSON schema, where CICS will generate your high-level data structure format, or from the structure to a JSON schema.

Using this feature, CICS will process an HTTP payload in JSON data format, and convert the data into the high-level language structure of a target CICS application, whether that application is in Common Business Oriented Language (COBOL), PL/I, C, or C++. The data will then be passed to the CICS application in either Channels and Containers or the communication area (COMMAREA).

When the application exits and control is returned, CICS will convert the output of the CICS application (in its high-level language structure) back to the JSON data type. This response is then sent back to the service requester through an HTTP payload.

The JSON to high-level language structure conversion service, and the high-level language structure to JSON conversion service, are also available through a LINKable program. CICS applications can use this LINKable program to call any external service expecting data in the JSON format. This enables your business applications to focus on business logic, and enables CICS to handle the burden of the data transformation.

When configured from a JSON schema, CICS can also be used to call your CICS applications in a RESTful architectural style. You are able to configure your JSON web services to call a number of different CICS programs based on the HTTP method used to make the call. The CICS programs then also receive further information, such as the query string that was used to make the RESTful request. This information can then be used in the logic of your CICS applications.

Liberty JSON/JAX-RS feature

IBM WebSphere Application Server Liberty profile (Liberty profile) is a dynamic profile that enables the server to provision only the features required by the applications deployed to the server. With CICS TS V5.1, a Liberty profile can run within a CICS Java virtual machine (JVM) server.

When using the Liberty profile in a CICS JVM server, you can configure the profile to enable the `jaxrs-1.1`. After enabling `jaxrs-1.1`, the feature provides support of the Java application programming interface (API) for RESTful web services (JAX-RS). JAX-RS is used for designing web services for the REST design pattern, using annotations to simplify the development and deployment of Java-based web service clients and endpoints.

Although further details of using the Liberty profile with JAX-RS enabled will not be covered in this book, information about the rapid implementation of an application that uses it can be found in the following IBM developerWorks® article:

https://www.ibm.com/developerworks/community/blogs/cicsdev/entry/writing_restful_web_services_using_cics_liberty_server_part_1?lang=en



CICS and IBM Worklight

In Chapter 2, “CICS use of mobile technologies” on page 11, you learned that Customer Information Control System (CICS) enables you to make new and existing mainframe applications available as web services using JavaScript Object Notation (JSON). It also provides support for the transformation between JSON and application data.

This chapter explains how IBM Worklight can be used in conjunction with CICS, and the advantages that this can have for your organization.

This chapter contains the following topics:

- ▶ 3.1, “Overview” on page 16
- ▶ 3.2, “Introduction to IBM Worklight” on page 16
- ▶ 3.3, “Using Worklight with CICS” on page 19

3.1 Overview

In cases where CICS is to act as a client of a JSON service, described in 4.2, “CICS as a client for JSON web services” on page 25, the interaction is usually with a partner company. In such scenarios, secured, direct interaction between CICS and other enterprise systems is appropriate. The same applies when using CICS as a service provider to other enterprise systems (for example, using CICS to provide data for an internal reporting tool).

If you want to use CICS as a service provider to support an application running on a mobile device, you should consider the benefits of using an intermediary system.

The example scenarios described later in this Redbooks publication demonstrate the simplicity of deploying a JSON-aware web service, issuing a request, and receiving a response. They show that, in principle, the CICS web services infrastructure exists to support interaction with mobile applications.

However, beyond providing proofs of concepts (POCs), you might find the absence of a full mobile application platform limiting, even though the functionality introduced in this chapter *does* enable a mobile device and a CICS application to communicate in a common language.

IBM Worklight provides a comprehensive platform for mobile application development, deployment, and governance. In particular, Worklight Server can act as an intermediary between mobile devices and CICS applications.

In the following section, an introduction to Worklight is provided, and Chapter 12, “IBM Worklight for CICS” on page 149, shows how this can be used as part of the example scenarios.

3.2 Introduction to IBM Worklight

Worklight provides an open and extensible mobile application platform. This platform enables organizations of all sizes to develop, run, and manage Hypertext Markup Language (HTML5), hybrid, and native mobile applications, as shown in Figure 3-1 on page 17.

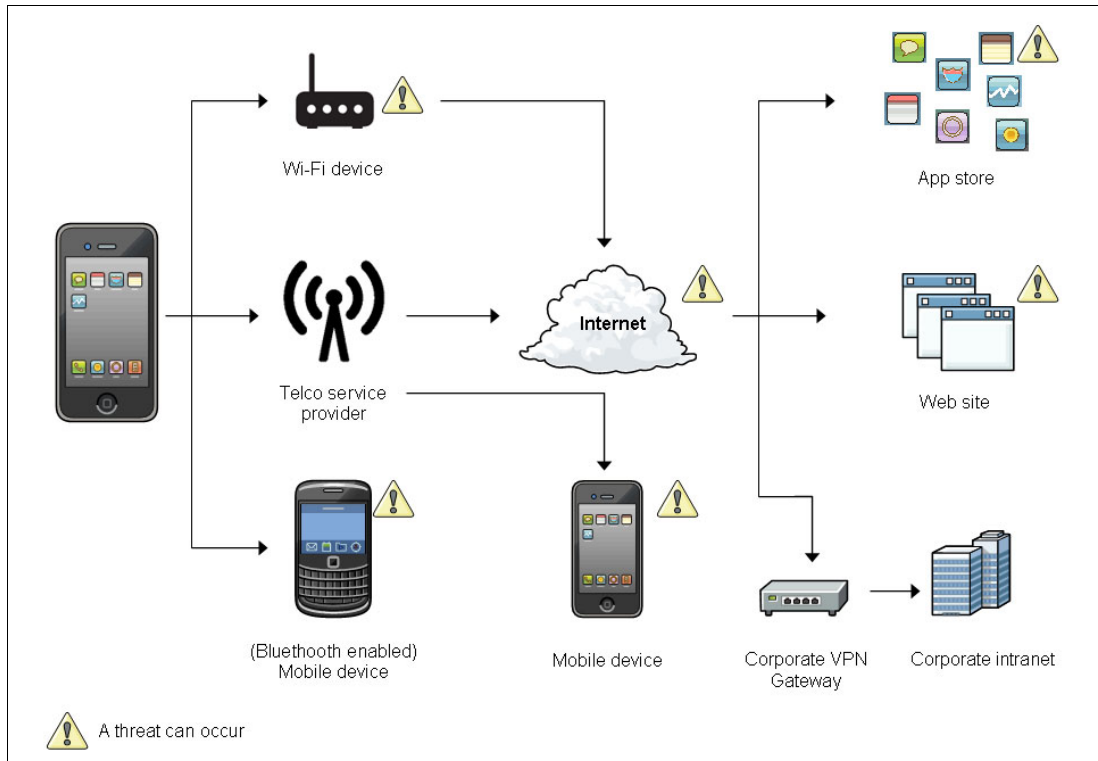


Figure 3-1 Flow of data transmission, and potential locations of exploitation

3.2.1 The Worklight platform

The Worklight platform consists of five main components: Worklight Studio, Worklight Server, Worklight device runtime components, Worklight Application Center, and Worklight Console.

Worklight Studio

Worklight Studio is an Eclipse-based integrated development environment (IDE), which can be used by your organization's mobile application developers for coding rich, cross-platform applications with a single, shared code base. This is achieved using standards-based technologies, and does not require the use of code translators or proprietary interpreters.

Worklight Studio can be used to build applications for most current mobile operating environments, including iOS, Android, BlackBerry, Microsoft Windows Phone, and Windows 8, as well as mobile web browsers. In addition, you can create applications for feature phones and embedded systems with Java Platform, Micro Edition (Java ME).

Developers can make use of third-party libraries and frameworks, such as Apache Cordova, Dojo Mobile, and jQuery Mobile. Furthermore, native code or JavaScript can be used to access mobile device application programming interfaces (APIs).

Application testing can be completed using the emulators provided with Worklight Studio. In addition, the ability to record, edit, and play back codeless test scripts on physical or emulated devices, reduces the time to value.

Worklight Studio can be installed as a new Eclipse instance, or into an existing Eclipse IDE.

Worklight Server

The Worklight Server is mobile-optimized middleware that provides a secure gateway between mobile applications, enterprise systems, and cloud-based services.

Multiple security mechanisms are supported, including integration with existing authentication and security methods. These features support the safeguarding of the device, application, and network layer.

The Worklight Server adapters add value to your mobile solution by providing server-side application code that connects to back-end systems and delivers data to and from mobile applications. Necessary server-side processing can be performed, reducing the need for processing on the mobile devices, or the modification of existing back-end systems and applications.

The Worklight Server adapters enable you to provide access to the transactional capabilities of CICS Transaction Server (CICS TS) for z/OS. Adapters are described further in the next section.

In addition, push notifications can be delivered to devices using a uniform cross-platform architecture, targeting users rather than devices. Services can be restricted by geolocation and short message service (SMS) notifications can be sent when the data network is unreliable.

Worklight device runtime components

Mobile applications deployed using IBM Worklight include client-side runtime APIs that embed server functionality in the applications. These APIs support the authentication between applications and Worklight Server, provide on-device encryption, and provide for the remote disablement of applications.

The APIs also provide a bridge between standard web technologies, such as HTML5, Cascading Style Sheets (CSS3), and JavaScript, with the native functions of the various mobile platforms.

The embedded code also assists with the push notification framework, and supports usage and event-based reports.

IBM Worklight Application Center

Employees within your organization might be familiar with the application stores of the various mobile platforms through personal use. The Worklight Application Center enables your company to set up an internal one-stop shop enterprise application store for the distribution of pre-release and production-ready applications.

Existing frameworks, such as Lightweight Directory Access Protocol (LDAP) and access control list (ACL), can be used to control distribution by department, job role, function, geographical area, or other topology. Your enterprise application store can be used to obtain feedback, feature requests, and enforce upgrades.

The distribution of pre-release mobile applications to development and test teams, in this way, accelerates the build-test-debug cycle.

Worklight Console

Worklight Server is administered through a graphical web-based interface, the Worklight Console. The Worklight Console supports the management of the server, adapters, and push services.

The following list notes additional features or actions possible when using the Worklight Console:

- ▶ Manage approved and rejected devices to control application installation.
- ▶ Control application versioning.
- ▶ Remotely disable applications by version or device type.
- ▶ Gain insight into the usage of instrumented applications.
- ▶ Produce user adoption and usage reports that can be processed by analytics platforms such as IBM Tealeaf®, IBM Cognos®, and IBM Coremetrics®.

3.2.2 Further reading

Use the following list of resources to obtain more information about Worklight:

- ▶ *Extending Your Business to Mobile Devices with IBM Worklight*, SG24-8117
- ▶ The IBM Mobile Foundation web page,
<http://www.ibm.com/software/mobile-solutions/>

3.3 Using Worklight with CICS

Worklight Server can act as a gateway between many mobile devices and CICS. Taking this approach, you protect your CICS systems by preventing direct access from mobile devices. You also benefit from the many other features of the Worklight platform concerning application development, deployment, and management.

3.3.1 Architecture overview

Figure 3-2 shows the positioning of CICS and Worklight in a mobile scenario.

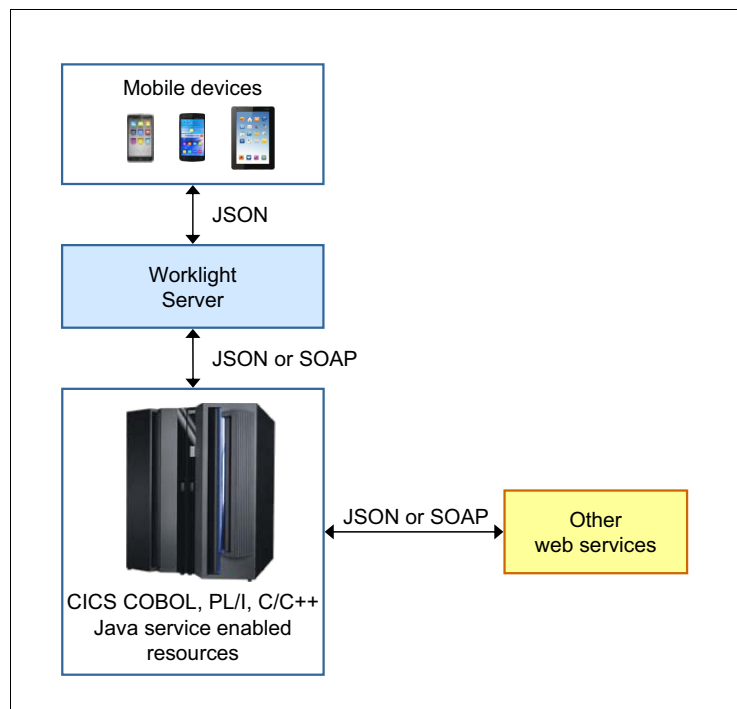


Figure 3-2 CICS to Worklight topology

Such a topology uses the Worklight Hypertext Transfer Protocol (HTTP) adapter for connectivity with either JSON or SOAP web services. Mobile applications communicate with the Worklight HTTP adapter, which sends requests to CICS on their behalf.

In addition to protecting your CICS systems from direct access, this approach has the following advantages:

- ▶ The adapter can call Representational State Transfer (REST)-conforming (RESTful) and SOAP web services provided by back-end systems. If necessary, the adapter can automatically convert between JSON and Extensible Markup Language (XML). Alternatively a developer can provide an Extensible Stylesheet Language (XSL) transformation, to explicitly define conversion.
- ▶ The adapter can modify the information returned from the back-end system.
- ▶ The adapter can cache frequently requested information.
- ▶ The adapter can issue requests to multiple back-end systems, then combine the retrieved information into one response back to the mobile application.
- ▶ Changes to the adapter are immediately available to all connected mobile devices. No updates to the mobile applications are required.

Chapter 6, “IBM Worklight configuration” on page 47, explains how to configure IBM Worklight to work with CICS. Reading that information helps in preparation for the example scenarios in Chapter 9, “Language structure to JSON schema scenario” on page 93 and Chapter 10, “JSON schema to language structure scenarios” on page 105.

Mobile applications present complicated security scenarios. Chapter 7, “Security and workload management” on page 69, poses questions for consideration before deploying a mobile application, and shows how CICS and IBM Worklight Server can be connected securely.



Patterns for JSON in CICS

There are a number of approaches and patterns for using JavaScript Object Notation (JSON) in CICS, with the capabilities of the Customer Information Control System (CICS) Transaction Server (CICS TS) Feature Pack for Mobile Extensions. These approaches are described in this chapter, and the advantages of each pattern are explained, giving examples of when they could be applied. This chapter covers the following topics:

- ▶ 4.1, “CICS as a JSON web service provider” on page 22
- ▶ 4.2, “CICS as a client for JSON web services” on page 25
- ▶ 4.3, “Handling JSON in other CICS applications” on page 27

4.1 CICS as a JSON web service provider

When CICS acts as a provider of a JSON web service, it receives incoming requests and calls CICS programs to process them. There are two main approaches to developing a JSON web service in CICS, depending on whether you start with an existing application or with a JSON message. In either case, you can use the CICS JSON assistant (batch utilities called DFHJS2LS and DFHLS2JS) to generate the necessary artifacts.

When you start with an existing JSON message, you can use either a Request-Response or Representational State Transfer (REST)-conforming (RESTful) pattern. However, if you want to reuse an existing application with the RESTful pattern, you must write a wrapper program. Figure 4-1 contrasts the two approaches, and they are explained in more detail in the following sections.

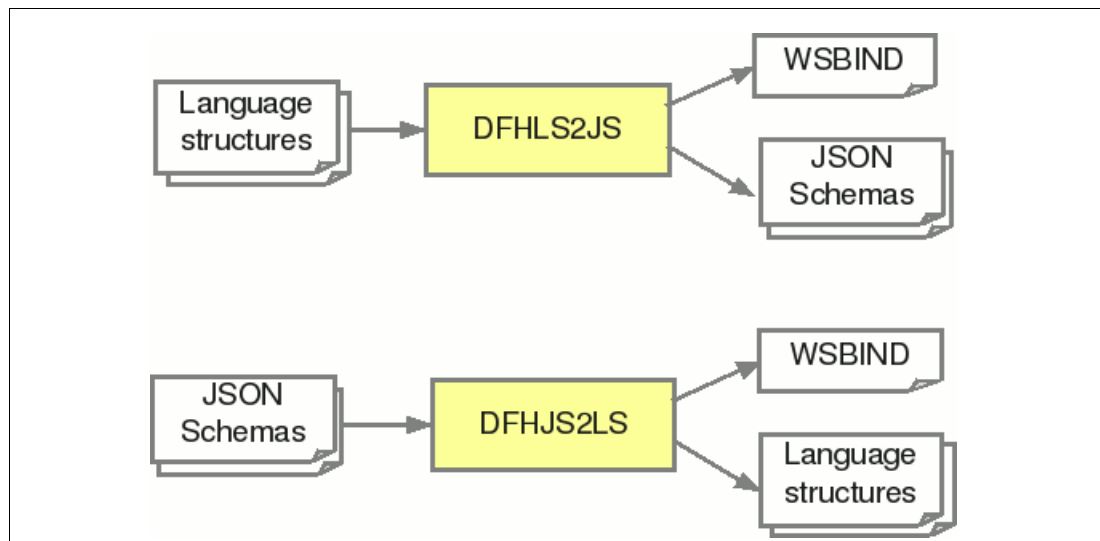


Figure 4-1 The two approaches for developing JSON web services

4.1.1 Starting with an existing application (bottom-up)

The *bottom-up approach* is used when you have an existing CICS application that you want to make available as a JSON web service. No changes are required to the application, and CICS handles the conversion between JSON and application data. The application does not need to have any knowledge that it is being started as a JSON web service.

This approach normally involves a relatively small implementation cost when compared to other approaches, and can also be low-risk, because the application remains unchanged. Chapter 9, “Language structure to JSON schema scenario” on page 93 describes a scenario that applies the bottom-up approach.

The interface to the application is described using high-level language structures, and the CICS JSON assistant generates JSON schemas describing the request-and-response messages. Then, a client application (which might be a mobile application) can be written using these schemas as a basis.

CICS transforms the incoming request to an application, and calls the application using either a channel and container or communication area (COMMAREA) interface. This action implies that the application is appropriately structured to separate business logic from presentation logic.

The described approach results in a web service interface that is closely coupled to the underlying application. All of the fields in the language structure will be present in the JSON messages (unless changes are made to the language structures specifically for service enablement), although they might not be required for all operations supported by the application.

These superfluous fields might result in larger-than-necessary payloads, and data formats that are not convenient for the client to provide. The bottom-up approach inherently creates a request-response style, activity-based interaction rather than a RESTful one.

4.1.2 Starting with an existing JSON interface (top-down)

The *top-down approach* is used when you have an existing JSON web service interface that you want to implement in CICS. You might be developing a new application, or you might have an existing application you want to adapt to a new interface. The interface might have been mandated by a partner company, an industry standard, or an existing client application.

This approach will always involve some development effort, and will require some part of the application to be aware of the JSON web service interface to a degree. Chapter 10, “JSON schema to language structure scenarios” on page 105 describes a scenario that applies the top-down approach.

The interface to the service is described using JSON schemas. As JSON web services are often documented in a less formal way, you might have to create a JSON schema. Then use the CICS JSON assistant to generate language structures.

If you are writing a new application, you can develop it based on these language structures. If you want to reuse an existing application, you can write a wrapper program based on the language structures (which program adapts data into a format acceptable to the existing application). At run time, CICS converts between JSON and application data described by the generated language structures.

CICS implements two patterns for JSON web services developed using this approach. You can choose to adopt either the Request-Response or RESTful patterns for your web service, depending on the interface to which you need to conform, or your business requirements.

Request-Response

The Request-Response pattern enables a remote procedure call-style interaction to be implemented, similar to that of SOAP web services. Services are activity-oriented, and typically provide one or more well-defined functions. The function to be performed, and its parameters, are normally identified as part of the payload.

Some application state might be maintained by the service. Every web service invocation involves a request message and a response message, and these might differ. Only the Hypertext Transfer Protocol (HTTP) POST method is supported by CICS when using the Request-Response pattern.

This pattern might be the closest match to the style of interface supported by a traditional CICS application, and therefore might require less development effort. It works well for applications where the emphasis is on the functions performed, rather than the resources they operate on. This pattern also works well for applications where the request and response messages differ.

For example, the Request-Response pattern might be a good fit for a banking application. In this case, the emphasis is on the actions being performed, such as deposits and withdrawals.

It wouldn't make sense to treat a withdrawal as a resource that is created and updated. Instead, the application would consist of services such as `getAccountBalance` and `transferFunds`, and the parameters (such as account numbers and amounts) would be carried in the request body.

RESTful

The RESTful pattern provided by CICS implements a pure form of the REST architectural style (described in 2.1, "REST" on page 12). A RESTful JSON web service operates on a single application-specific resource, which is normally identified by the Uniform Resource Identifier (URI). A single message format describes this resource, and is used for either the HTTP request or response, depending on the function. The function performed on the resource is determined by the HTTP method.

A RESTful web service provider program (which might be a wrapper program) must perform the following tasks:

- ▶ Identify the resource from the URI.
A RESTful web service request relates to a specific resource that is normally identified by the URI. Typically, a URIMAP with a wildcard will be used so that CICS calls the web service for any instance of a given resource. The application must extract the resource identifier from the URI. CICS provides several containers with fragments of the URI to help with the identification.
- ▶ Check the HTTP method to determine what function to perform.
CICS puts the HTTP method in a container, which the application must read and perform the corresponding function. The function is application-dependent, and might involve linking to other business logic. A service does not need to support each method, and you can specify which methods your service accepts when using the JSON assistants. CICS validates that the method in the request is supported by the service before calling the program.
- ▶ Return an appropriate response.
If the method requires a response, the application can return data that CICS will transform to JSON. Otherwise, the application can set the HTTP status to indicate success or error. You can also choose to send a custom HTTP response body directly.

For more information about these tasks, see "Creating a RESTful web service provider application" in the CICS TS Feature Pack for Mobile Extensions Information Center. For CICS TS 5.1, see the following website:

http://pic.dhe.ibm.com/infocenter/cicsts/v5r1/topic/com.ibm.cics.ts.mobileextensions.doc/tasks/restws_create_provider.html

Figure 4-2 on page 25 shows how a wrapper program can be used to perform these tasks, making existing business logic available as a RESTful JSON web service.

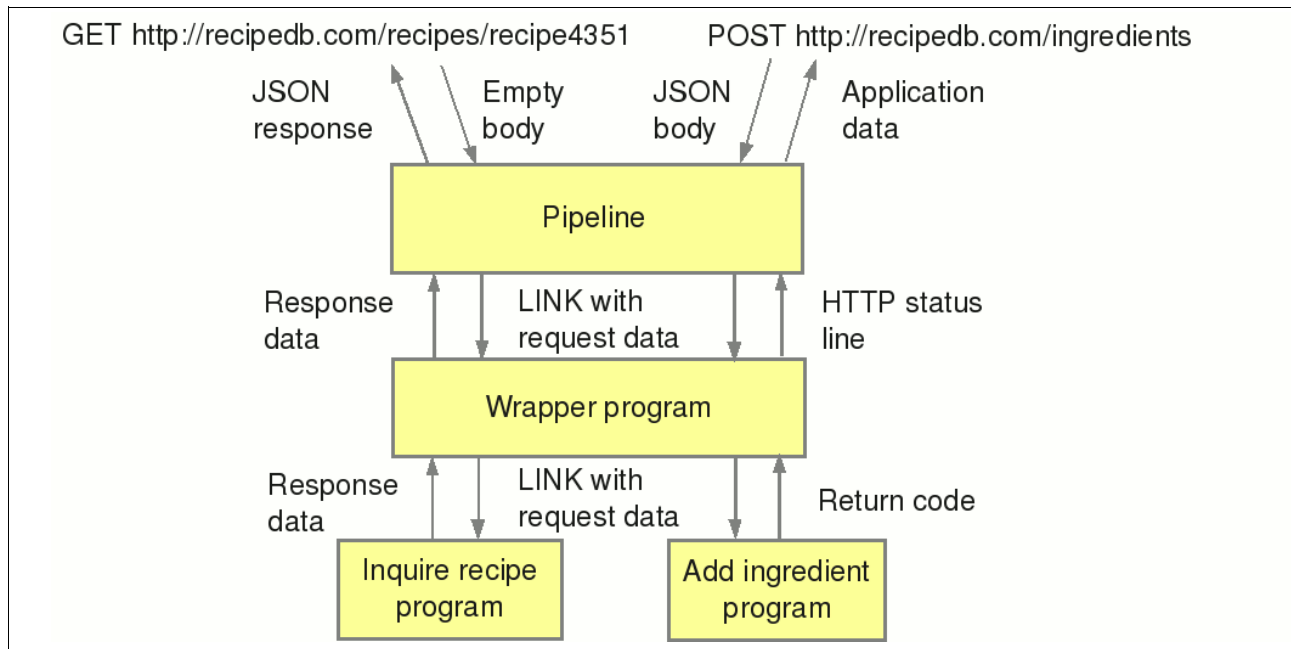


Figure 4-2 Using a wrapper program to make an existing application available as a RESTful JSON web service

Adopting the RESTful pattern requires at least some degree of application development to extract the relevant information from the request and convert it to a form usable by the business logic. For an application to be truly RESTful, it should be designed using the pattern from the ground up.

However, some benefits could be gained from adopting a RESTful pattern for the interface, even if the application itself does not fully implement REST. RESTful web services are most suited to applications where the focus is on the resources, and the functions are a good match with the set of HTTP methods.

For example, the RESTful pattern would be a good fit for an application that provides an online recipe book. It might have resources representing recipes, ingredients, and cooking techniques.

You would implement a JSON web service in CICS for each of these resources. Information about these resources might be retrieved individually (a recipe for sponge cake) or as collections (all of the ingredients for rice pudding). New recipes can be added, existing ones modified, and redundant ones deleted. However, the emphasis is on the resources and not the operations.

4.2 CICS as a client for JSON web services

In addition to making applications available as JSON web services, CICS can operate as a client of other JSON web services. This capability can be used to integrate the functionality provided by other JSON web services into the business logic of a CICS application.

4.2.1 Integrating other JSON web services into your CICS application

When designing a CICS application, you might want to use functionality provided by another JSON web service. This service could be hosted in CICS, or could be hosted on another platform.

For example, when processing a credit card application, you might want to obtain a credit score from a partner company. The partner company might make this functionality available as a JSON web service. The CICS TS Feature Pack for Mobile Extensions V1.0 provides the capability to convert between application data and JSON when starting such a service. This enables you to incorporate the credit scoring function into the business logic of your application.

Another possible scenario involves a CICS application acting as an aggregator of information from multiple services. For example, an insurance broker might implement an application in CICS to find the most competitive quote for a customer. The application would aggregate quotes from many insurance providers and compare them. Some insurance providers might also make their quoting applications available as JSON web services.

4.2.2 How CICS supports acting as a client for JSON web services

Writing a CICS application that acts as a client of JSON web service involves using a linkable interface to transform between application data, JSON, and the CICS WEB API commands to communicate with the service. If you are already familiar with how CICS supports acting as a client for SOAP web services, you should be aware that this approach is somewhat different.

When writing a client for a JSON web service, both the **INVOKE SERVICE** API command and the pipeline are not used. The approach used for transforming between JSON and application data, in this case, is somewhat similar to the **TRANSFORM** application programming interface (API) used for Extensible Markup Language (XML).

Developing an application that acts as a client of a JSON web service begins by defining the interface to the service. In most cases, the interface will already exist, defined by the party providing the service. CICS requires a JSON schema describing the interface to the service, so if one does not already exist you will need to create it.

Alternatively, if the service does not yet exist, you can start with a language structure to define the interface to the application. After you have obtained either a JSON schema or a language structure, run the CICS JSON assistants. This creates either a language structure or a JSON schema, and a CICS bundle that contains the mapping that will be used at run time to transform between application data and JSON.

You can then write a CICS application that uses the CICS WEB API commands to connect to the JSON web service. Depending on the interface to the service, you might need to send a JSON request, or the request might be encoded in the URI of the service. If you need to send JSON, you can use the linkable interface to transform your application data to JSON. Then, read the response from the remote service and, if necessary, use the linkable interface to transform the response to application data.

The linkable interface used to transform between application data and JSON consists of a transformer program provided by CICS and a designed set of containers that must be populated by your application. You use an **EXEC CICS LINK PROGRAM** command to call the transformer, and data is returned in containers.

For an example of how to write a client for a JSON web service, see Chapter 11, “Developing a simple JSON web service client application” on page 127.

4.3 Handling JSON in other CICS applications

Using JSON in CICS applications is not limited to web services. The linkable interface for transforming between JSON and application data can process JSON from any source, and for any purpose. This opens a wide range of possible uses of JSON in your CICS applications, whether you are reusing existing assets or creating new ones. The following list notes some examples of other uses for JSON:

- ▶ Sending and receiving JSON over transport protocols other than HTTP, such as WebSphere MQ or raw sockets.
- ▶ Interacting with a JSON data store.
- ▶ Interoperating with applications written in server-side Javascript, where JSON is the data interchange format of choice.
- ▶ Implementing complex web service interactions that cannot be implemented using CICS JSON web service support. These might include services that support several message types, or where you want to mix RESTful and Request-Response patterns.

For more information about using the linkable interface to transform JSON, see 11.1.3, “The linkable interface for transforming JSON” on page 129 and the “Transforming application data and JSON using the linkable interface” topic in the CICS TS Feature Pack for Mobile Extensions Information Center. For CICS TS 5.1, see the following website:

http://pic.dhe.ibm.com/infocenter/cicsts/v5r1/topic/com.ibm.cics.ts.mobileextensions.doc/tasks/transforming_json_linkable_interface.html

Part 2



Setup and configuration

This part provides information about the setup and configuration of Customer Information Control System (CICS) for the example scenarios, IBM Worklight Server, Security, workload management, and problem determination.



Configuring CICS for the example scenarios

This chapter describes how to prepare your Customer Information Control System (CICS) environment for the scenarios presented in Part 3, “Application development and scenarios” on page 91. The chapter further provides information about each resource definition and its role when CICS handles or makes a request.

The scenarios are described in the following list:

- ▶ Taking an existing CICS application and making it available as a JavaScript Object Notation (JSON) web service, as introduced in 4.1.1, “Starting with an existing application (bottom-up)” on page 22. This scenario can be found in Chapter 9, “Language structure to JSON schema scenario” on page 93.
- ▶ Creating a new CICS application based upon a JSON schema, introduced in 4.1.2, “Starting with an existing JSON interface (top-down)” on page 23. This scenario is described in Chapter 10, “JSON schema to language structure scenarios” on page 105, where CICS acts as a service provider.
- ▶ Using CICS as a JSON client, as explained in 4.2, “CICS as a client for JSON web services” on page 25. This scenario is found in Chapter 11, “Developing a simple JSON web service client application” on page 127.

As described in Chapter 2, “CICS use of mobile technologies” on page 11, the JSON web services functionality of CICS is built upon the established web services infrastructure. The material in this chapter will be familiar to those with experience using SOAP web services.

This chapter contains the following topics:

- ▶ 5.1, “Comparison with SOAP web services” on page 32
- ▶ 5.2, “CICS as a service provider” on page 32
- ▶ 5.3, “CICS as a JSON client” on page 46

5.1 Comparison with SOAP web services

To provide support for web services requests using JSON, and the conversion between JSON and application data, internal changes were made to the CICS web services pipeline. The externals, and therefore the CICS resources created in this chapter, are the same as are required for SOAP web services.

At the core of the JSON support in CICS are the JSON assistants. Most relevant to application developers, the assistants consist of job control language (JCL)-based tools used for preparing applications for use as JSON web services. The DFHLS2JS JCL procedure is provided for converting a high-level language structure into a JSON schema. In Chapter 9, “Language structure to JSON schema scenario” on page 93, you see how this can be used to make an existing CICS application available through a web service.

Conversely, the DFHJS2LS JCL procedure also provides for converting a JSON schema into a high-level language structure, suitable for use with a new CICS application or Representational State Transfer (REST)-conforming (RESTful) interface. This utility is demonstrated in Chapter 10, “JSON schema to language structure scenarios” on page 105.

The JSON assistants are equivalent to the web service assistants used for conversion between high-level languages and Web Services Description Language (WSDL). The examples in this IBM Redbooks publication convert JSON schema to and from Common Business Oriented Language (COBOL). In addition to COBOL, the following languages can be converted to and from JSON schemas using the JSON assistants:

- ▶ C
- ▶ C++
- ▶ PL/I

5.2 CICS as a service provider

This section explains the resources used by CICS to process a web services request. This is followed by the steps required to configure CICS, in preparation for the deployment of the services presented in Chapter 9, “Language structure to JSON schema scenario” on page 93, and Chapter 10, “JSON schema to language structure scenarios” on page 105.

5.2.1 How CICS processes a request

Figure 5-1 on page 33 depicts how CICS processes a web services request with JSON data.

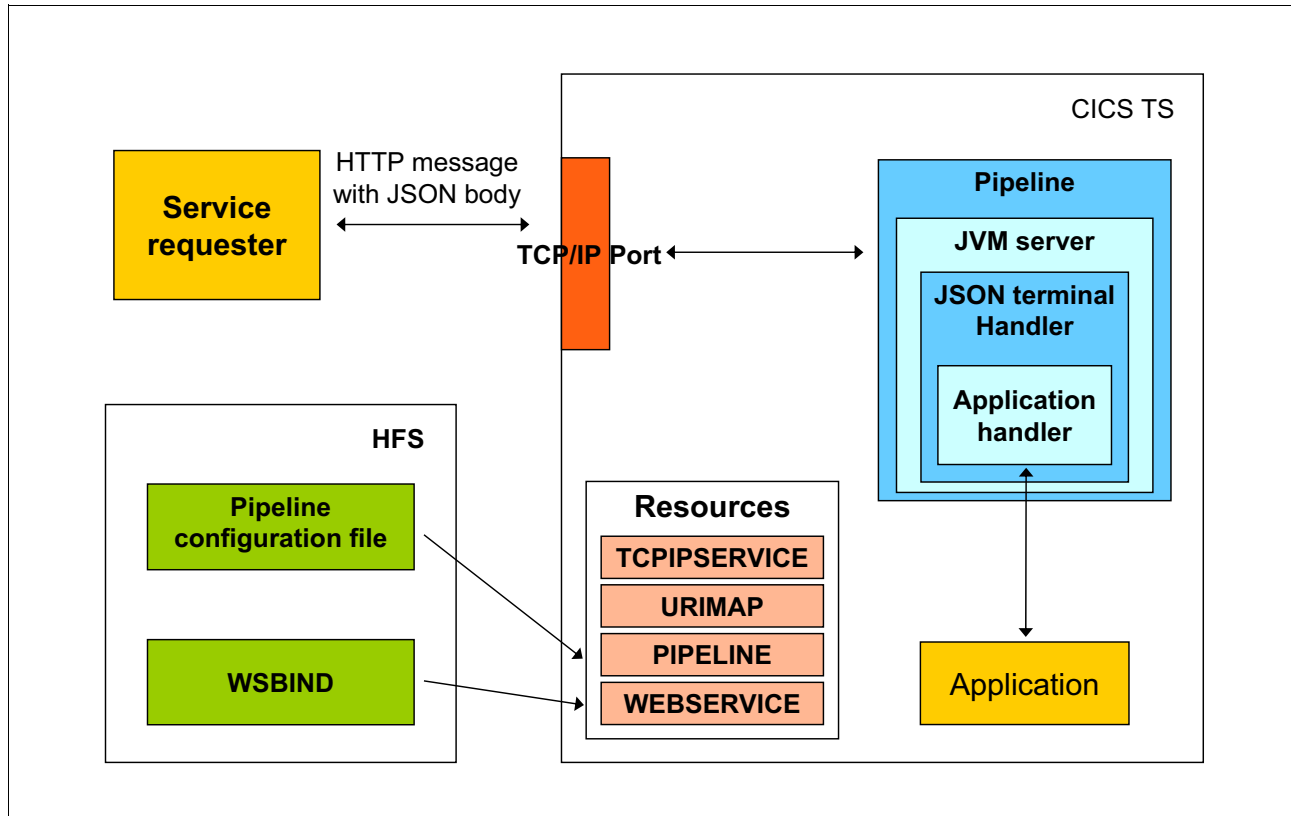


Figure 5-1 Architecture of JSON supports in CICS

The following procedure shows how the JSON data is used by CICS to process the web services request:

1. A request is made by the Hypertext Transfer Protocol (HTTP) protocol on a port opened by an installed Transmission Control Protocol/Internet Protocol (TCP/IP) TCPIP SERVICE definition. The port is monitored by the CICS sockets listener (CSOL) transaction. CSOL attaches the transaction specified in the TRANSACTION attribute of the TCPIP SERVICE definition. Usually, this is the CICS web attach transaction **CWXN**.
2. **CWXN** matches the incoming request with a Uniform Resource Identifier (URI) URIMAP by scanning all URIMAP definitions for one that has its USAGE attribute set to PIPELINE and its PATH attribute set to the URI found in the request. The URIMAP definition indicates the PIPELINE and WEBSERVICE definitions to be used, and the TRANSACTION that should be attached to process the PIPELINE. The transaction is usually **CPIH**.
3. As with SOAP web services, the WEBSERVICE definition points to a WSBIND file. This will be used later for data transformation between JSON and application data. The WSBIND file will have been created using the JSON Assistant.
4. Pipeline processing then takes place, passing the request through any defined handlers.
5. The JSON terminal handler and CICS application handler are called. The JSON data is converted into application data, using the language structure description in the WSBIND file. The application handler links to the application program. Note, the application program might be run in a different CICS region.
6. Finally, the PROGRAM output is captured and sent back to the requester as an HTTP response with JSON data.

Note: The pipeline processing described previously runs within a Java virtual machine (JVM) server environment. If configured appropriately, some of the processing that takes place within the pipeline is eligible for offloading onto zSeries Application Assist Processors (zAAPs).

5.2.2 How to configure CICS as a service provider

To prepare the CICS region for the scenarios described in Chapter 9, “Language structure to JSON schema scenario” on page 93, and Chapter 10, “JSON schema to language structure scenarios” on page 105, the following tasks must be completed:

1. Define and install a TCPIPSERVICE.
2. Install a JVMSERVER, configured for use by a PIPELINE.
3. Define and install a PIPELINE.

Important: You must specify **TCPIP=YES** in your CICS region’s system initialization parameters to activate CICS TCP/IP services.

Later chapters show how to create the URIMAP and WEBSERVICE resources automatically using a CICS PIPELINE scan.

Alternatively, you can create these resources yourself. This gives more control, but requires additional resource management.

This book shows how to complete these tasks, which would typically be undertaken by a CICS system programmer, using IBM CICS Explorer 5.1.1.

Note: For instructions on how to connect CICS Explorer to a CICS region or IBM CICSplex® SM system, see the “Configuring the CICS Explorer” topic in the CICS Transaction Server (CICS TS) Information Center.

Defining and installing a TCPIPSERVICE

Begin by defining a TCPIPSERVICE in the CICS region. To do this, first open the TCP/IP Service Definitions view from the **Definitions** menu:

1. Right-click an unpopulated row and click **New**, as shown in Figure 5-2.

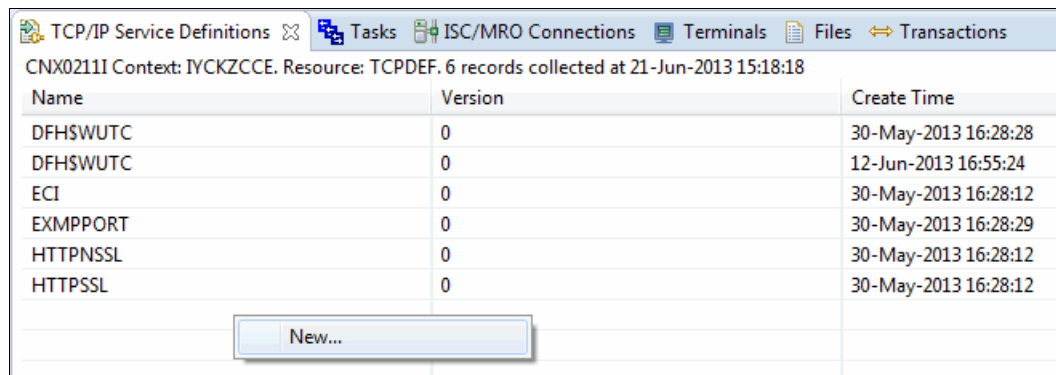


Figure 5-2 Right-click an unpopulated row for the option to create a new resource definition

2. The New TCP/IP Service Definition window opens. Provide the following parameters:
 - a. In the **Resource/CSD Group** field, enter the group in which you want to place the new TCPIP SERVICE.
 - b. In the **Name** field, enter an appropriate name.
 - c. In the **Description** field, describe the TCPIP SERVICE that you are creating.
 - d. Finally, enter the port that you want CICS to accept incoming requests through in the **Port Number** field.

Figure 5-3 shows the New TCP/IP Service Definition window and entry fields.

The screenshot shows a Windows-style dialog box titled "New TCP/IP Service Definition". The main heading inside is "Create TCP/IP Service Definition". The fields are as follows:

- CICSplex: IYCKZCCE
- Region (CSD): IYCKZCCE
- Resource/CSD Group: GENAMOBL
- Name: GENAMOBL
- Description: A TCP/IP Service for the GENAPP applications.
- Port Number: 30661
- Open editor

At the bottom right, there are "Finish" and "Cancel" buttons. A help icon (?) is located at the bottom left.

Figure 5-3 Entering the attributes of a new TCPIP SERVICE

3. Click **Finish**. Click **Open editor** and a new editor will open, as shown in Figure 5-4. This view shows that the newly created TCPIP SERVICE will use the HTTP protocol.

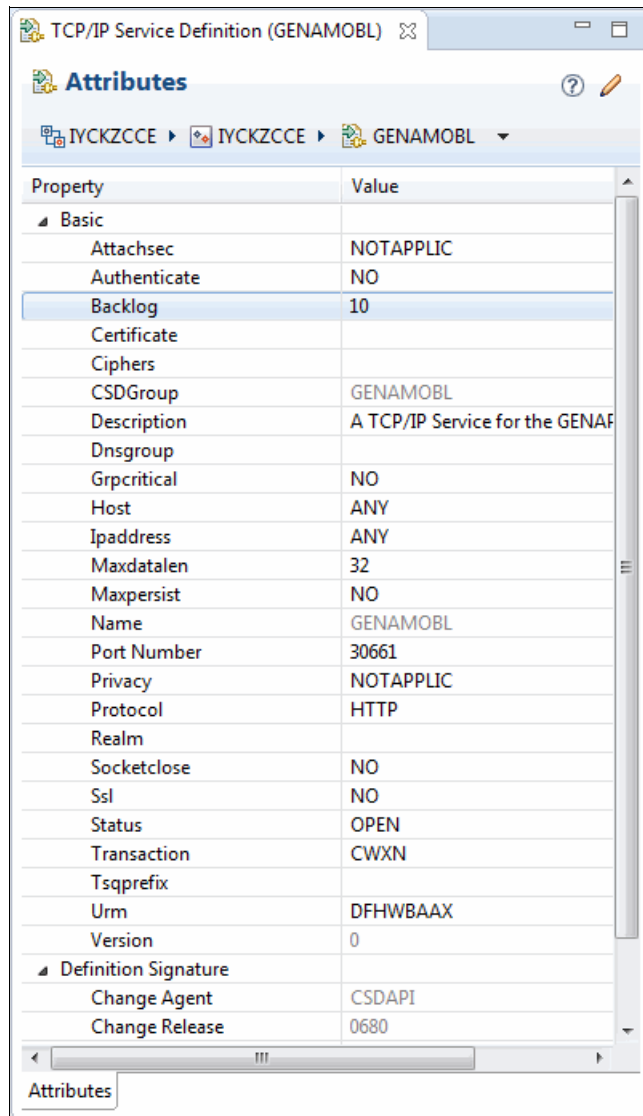


Figure 5-4 Viewing the attributes of a TCPIP SERVICE

The *Backlog* attribute specifies the maximum number of inbound TCP/IP connection requests that can be queued in TCP/IP for CICS processing. If this number is reached, TCP/IP will reject additional connection requests. For the purposes of this chapter’s example scenarios, this attribute is set to 10.

The *Maxdatalen* attribute specifies the maximum data length that can be received by CICS via HTTP. The examples in this scenario are left unchanged, at 32,000. However, before deploying a web service in a production environment, consider the amount of data that you expect that service to receive, and set an appropriate limit. This helps to guard against denial of service attacks using large amounts of data.

The attributes also indicate that no security is currently configured. Although this would be undesirable for a production environment, for the example scenarios in a testing environment this is acceptable. See Chapter 7, “Security and workload management” on page 69 for information about how you can secure TCPIP SERVICES.

Note: For more information about TCPIP SERVICE attributes see the “TCPIP SERVICE resources” topic in the CICS TS Information Center.

The new TCPIP SERVICE is also displayed in the TCP/IP Service Definitions view.

4. Right-click the new TCPIP SERVICE and click **Install**, as shown in Figure 5-5.

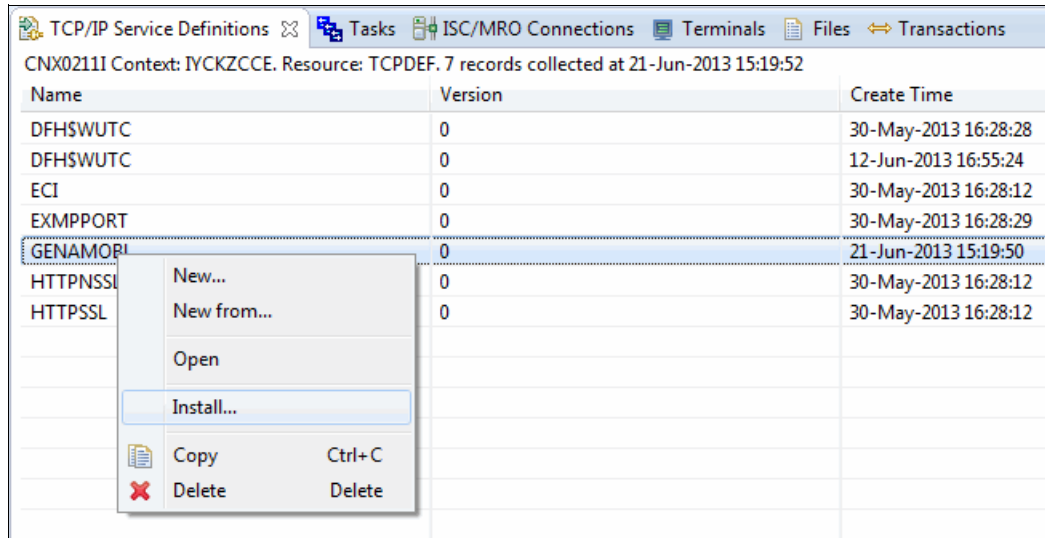


Figure 5-5 Preparing to install a TCPIP SERVICE

A Perform Operation window will open, as shown in Figure 5-6.

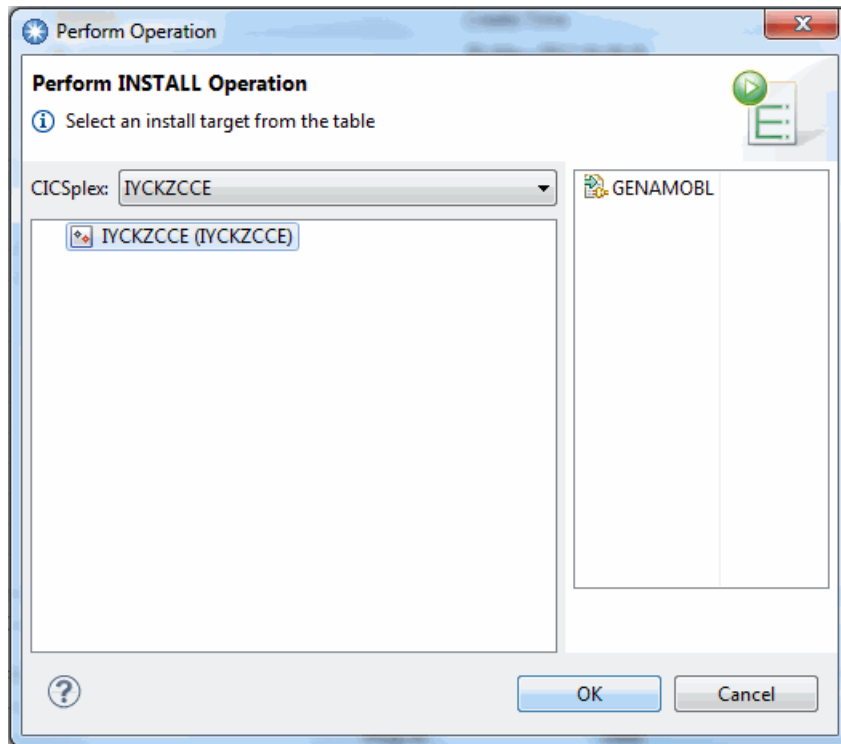


Figure 5-6 Performing an install of a TCPIP SERVICE

5. Select the CICS system where you want to install the TCPIP SERVICE, and then click **OK**. The window will close if the operation was performed successfully. If an error occurred, perform the necessary steps to correct the problem and repeat the operation.
6. Open the TCP/IP Services Operations view by clicking **Operations** → **TCP/IP Services**. From this view, illustrated in Figure 5-7, you can see that the TCPIP SERVICE, created for the GENAPP scenarios, was installed and has a Service Status of OPEN.

Region	Name	Port	Service Status
IYCKZCCE	DFHSWUTC	30660	OPEN
IYCKZCCE	GENAMOBL	30661	OPEN

Figure 5-7 View of installed TCP/IP Services

If you were to open a web browser and send an HTTP request to the address of your CICS system using the port number specified in your TCPIP SERVICE, you will receive an HTTP 404 response. This indicates that CICS cannot locate a resource corresponding to the URI specified.

Installing a JVM SERVER, configured for use by a PIPELINE

To install a JVM SERVER, enabled for use by a PIPELINE, requires a JVM profile that has the `JAVA_PIPELINE=YES` option specified.

An example JVM SERVER resource definition, called `DFH$AXIS`, is supplied with CICS, in group `DFH$AXIS`. The resource `DFH$AXIS` uses the supplied JVM profile, `DFHJVMAX`, which specifies `JAVA_PIPELINE=YES`.

`DFHJVMAX` can be found in the `/JVMPProfiles` directory of your CICS installation.

Perform the following instructions to install a JVM SERVER resource:

1. Copy `DFHJVMAX` to a different directory, which should be set as the `JVMPROFILEDIR` system initialization parameter of your CICS system. You should also copy `DFH$AXIS` to a new group.

Important: Using definitions and files supplied with CICS in their default locations is not suggested, as they could be updated by corrective maintenance.

2. To create a copy of `DFH$AXIS`, click **Definitions** → **JVM Server Definitions**. The list will be populated by the JVM SERVER resource definitions within your CICS Explorer context. If necessary, click the refresh button in the upper right part of the view.
3. Right-click `DFH$AXIS` and click **New from**, as shown in Figure 5-8 on page 39.

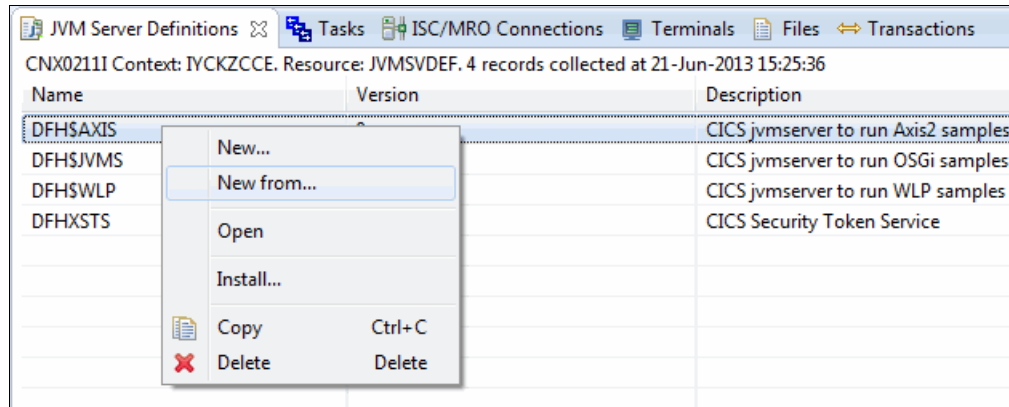


Figure 5-8 Creating a new JVMSERVER resource from an existing definition

4. When the new JVM Server Definition window opens, as shown in Figure 5-9, modify the following attributes:
 - a. Change the **Resource/CSD Group** to a different location.
 - b. Using the **Name** field, change the name of the JVMSERVER.
 - c. Alter the **Description** to a more appropriate value.

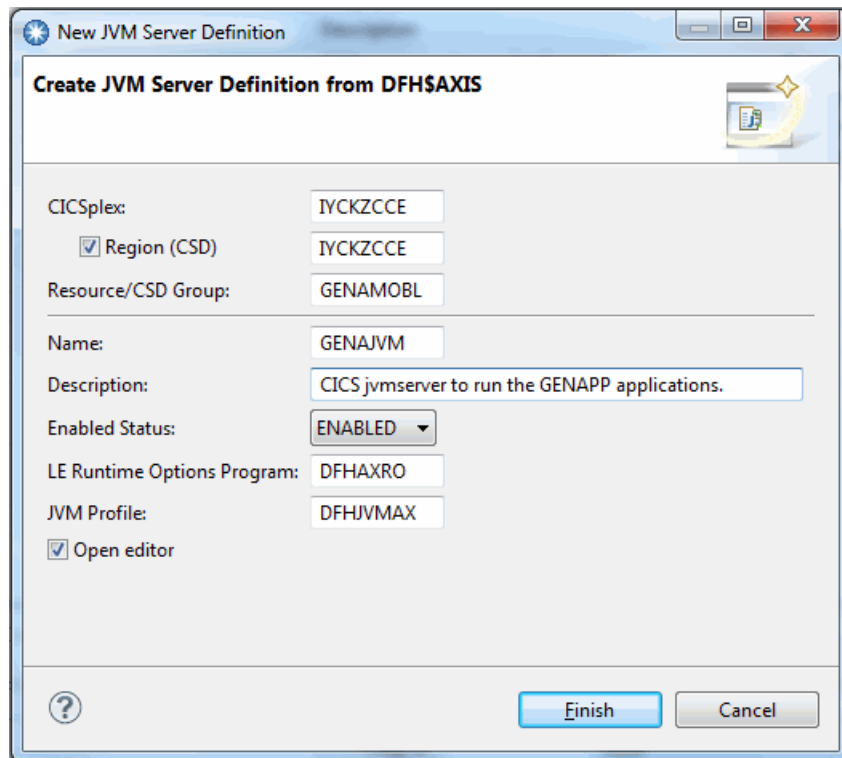


Figure 5-9 Defining a JVM server

- Click **Finish**. Your new JVMSERVER definition will be listed in the JVM Server Definitions view. Right-click the new definition and click **Install**, as shown in Figure 5-10.

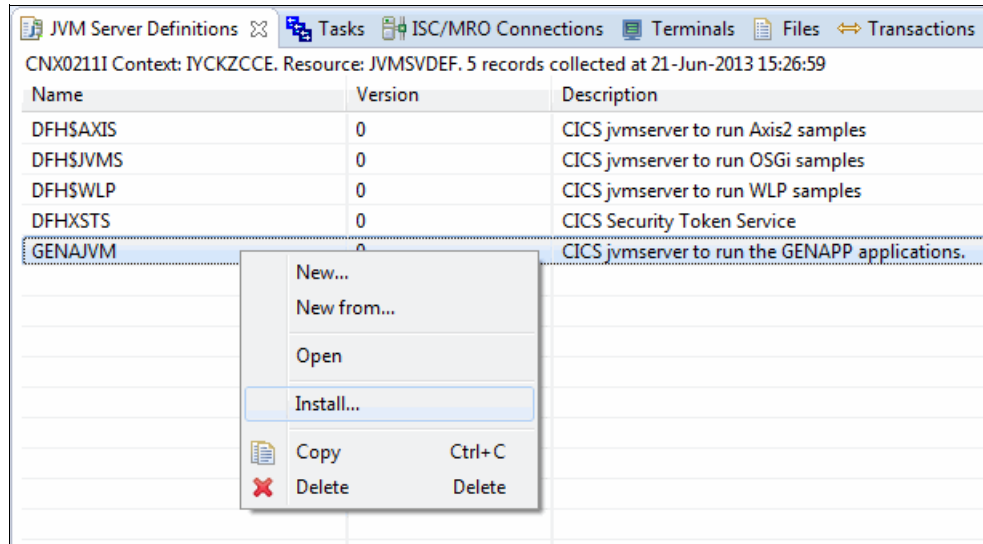


Figure 5-10 About to install a JVM server

- When the Perform Operation window opens, select your CICS system and click **OK**. When the operation is complete, you can view your installed JVM servers by clicking **Operations** → **Java** → **JVM Servers**. The JVM server should have an Enable Status of **ENABLED**, as shown in Figure 5-11.

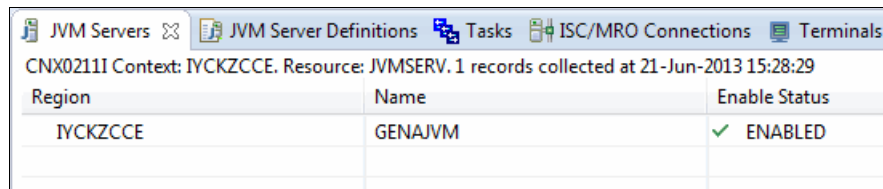


Figure 5-11 The view of installed JVM servers

You can use one JVM server as the runtime environment for multiple Java pipelines. Note that each task is attached to a JVM thread using a T8 task control block (TCB), with the total number of threads limited by the JVMSERVER resource’s THREADLIMIT attribute. In addition, there is a limit on the number of T8 TCBs that can exist in a CICS region across all JVM servers. For further information, see the topic about managing the thread limit of JVM servers, found in the CICS TS Information Center appropriate to your release version of CICS TS.

You can read about planning for large workloads in 7.4, “Workload management overview” on page 80.

Defining and installing a PIPELINE

To complete preparation for the example scenarios described in Chapter 9, “Language structure to JSON schema scenario” on page 93, and Chapter 10, “JSON schema to language structure scenarios” on page 105, a PIPELINE is required.

Before creating a PIPELINE resource definition, first prepare z/OS File System (zFS) directories for the following items:

- ▶ The pipeline configuration file. This is an XML file that describes handler programs that CICS starts when it processes the pipeline. For JSON web services, this will be the CICS JSON terminal handler and application handler, in addition to the JVM server to be used for pipeline processing.

Note that the same directory can be used for many pipeline configuration files, and a pipeline configuration file can be used by many pipelines.

- ▶ The WSDIR. This directory, also known as the *pickup directory*, can be used for installing WSBIND files. You will learn more about this in Chapter 9, “Language structure to JSON schema scenario” on page 93, and Chapter 10, “JSON schema to language structure scenarios” on page 105. You should have one WSDIR per pipeline.
- ▶ The shel f directory. This is a directory used by CICS to store WSBIND files. It is used for recovery across a warm restart of CICS.

The CICS Transaction Server for z/OS Feature Pack for Mobile Extensions V1.0 provides a sample pipeline configuration file for JSON pipelines in the `/usr/lpp/cicsts/mobilefp/samples/pipelines` directory, where `/usr/lpp/cicsts/mobilefp` is the feature pack installation directory.

The scenario uses this configuration file, named `jsonjavaprovider.xml`, for the examples. As with the JVM profile, copy the configuration file to a different directory. Further, update the JVM server name specified in the configuration file to match that of the JVMSERVER defined previously.

Note: A single PIPELINE resource can be used for multiple JSON or multiple SOAP web services. However, you cannot use a PIPELINE for both types of web service.

By default, CICS will use `/var/cicsts` as the shel f directory.

The next step for creating your PIPELINE is to create a suitable pickup directory. You can create directories using the z/OS perspective of CICS Explorer, using Secure Shell (SSH), or the `Udlist` utility of IBM Interactive System Productivity Facility (ISPF).

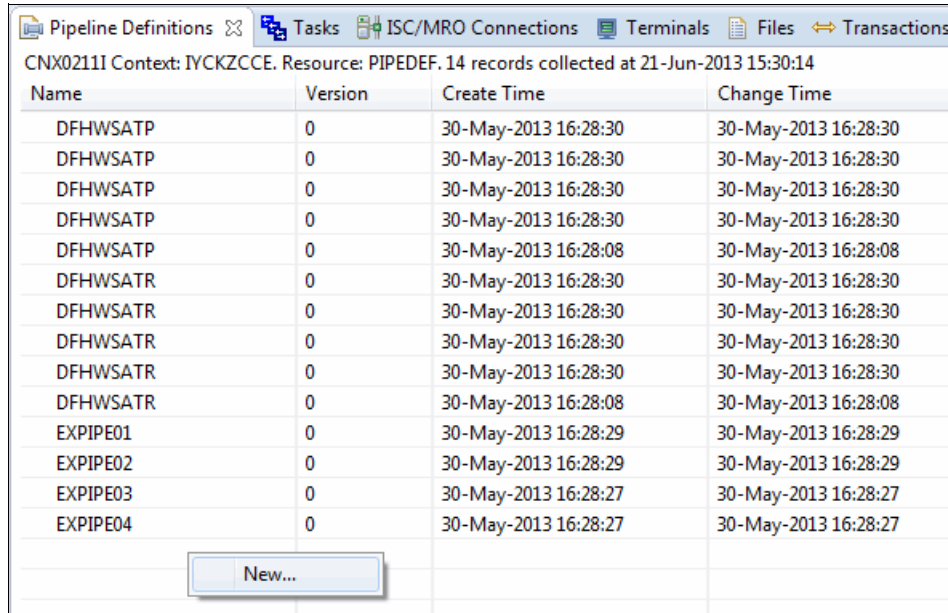
Optionally, create a shel f directory if you do not want to use the default.

Important: Ensure that CICS has permission to at least read the pickup directory and the directory used to store pipeline configuration files. CICS must also have permission to read, write, and create sub-directories within the shel f directory.

Perform the following steps to define and install a PIPELINE resource:

1. After your directories have been prepared, in CICS Explorer, click **Definitions** → **Pipeline Definitions**.

2. Right-click **New**, as shown in Figure 5-12.



The screenshot shows the CICS Explorer interface with a table of pipeline definitions. The table has four columns: Name, Version, Create Time, and Change Time. A 'New...' button is highlighted at the bottom of the table.

Name	Version	Create Time	Change Time
DFHWSATP	0	30-May-2013 16:28:30	30-May-2013 16:28:30
DFHWSATP	0	30-May-2013 16:28:30	30-May-2013 16:28:30
DFHWSATP	0	30-May-2013 16:28:30	30-May-2013 16:28:30
DFHWSATP	0	30-May-2013 16:28:30	30-May-2013 16:28:30
DFHWSATP	0	30-May-2013 16:28:08	30-May-2013 16:28:08
DFHWSATR	0	30-May-2013 16:28:30	30-May-2013 16:28:30
DFHWSATR	0	30-May-2013 16:28:30	30-May-2013 16:28:30
DFHWSATR	0	30-May-2013 16:28:30	30-May-2013 16:28:30
DFHWSATR	0	30-May-2013 16:28:30	30-May-2013 16:28:30
DFHWSATR	0	30-May-2013 16:28:08	30-May-2013 16:28:08
EXPIPE01	0	30-May-2013 16:28:29	30-May-2013 16:28:29
EXPIPE02	0	30-May-2013 16:28:29	30-May-2013 16:28:29
EXPIPE03	0	30-May-2013 16:28:27	30-May-2013 16:28:27
EXPIPE04	0	30-May-2013 16:28:27	30-May-2013 16:28:27

Figure 5-12 How to create a new PIPELINE using CICS Explorer

3. A New Pipeline Definition window will open. Complete the fields as noted in the following sub-steps:
 - a. In the **Resource/CSD Group** field, enter the resource group in your CICS system definition data set (CSD) in which you want the new PIPELINE placed.
 - b. In the **Name** field, enter an appropriate name.
 - c. In the **Description** field, describe the PIPELINE you are creating.
 - d. Next, in the **Configuration File** field enter the path to the pipeline configuration file, within z/OS UNIX.

See Figure 5-13 on page 43 for an illustration of the New Pipeline Definition window.

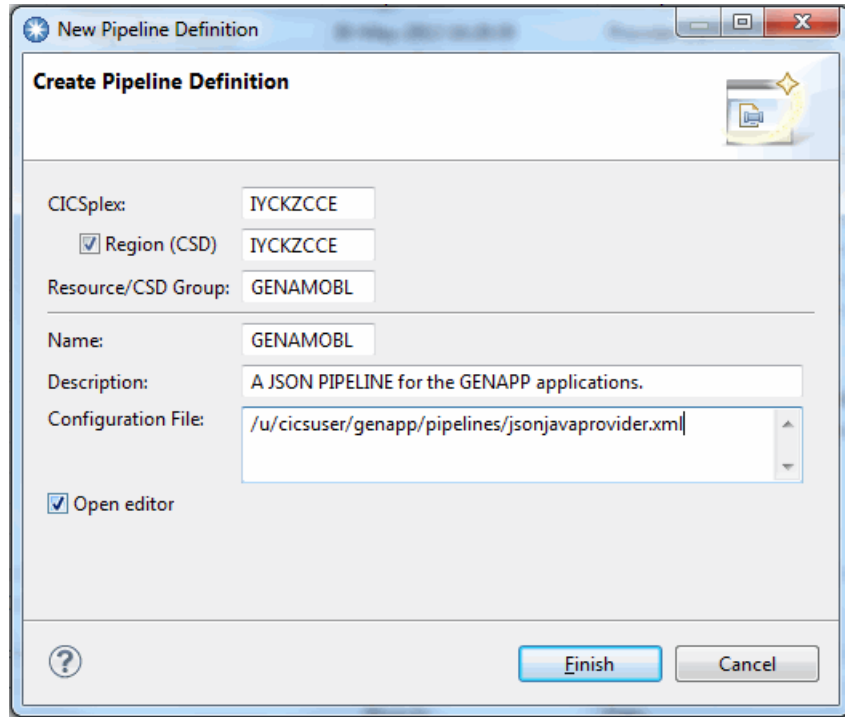


Figure 5-13 Defining a PIPELINE resource

4. Click **Finish**.

5. Select **Open editor**. The Pipeline Definition view will open to the right of CICS Explorer, as shown in Figure 5-14. Enter the location of your pickup directory in the **Name of a directory (shelf) for WSBind files** field in the hierarchical file system (HFS) Details area.

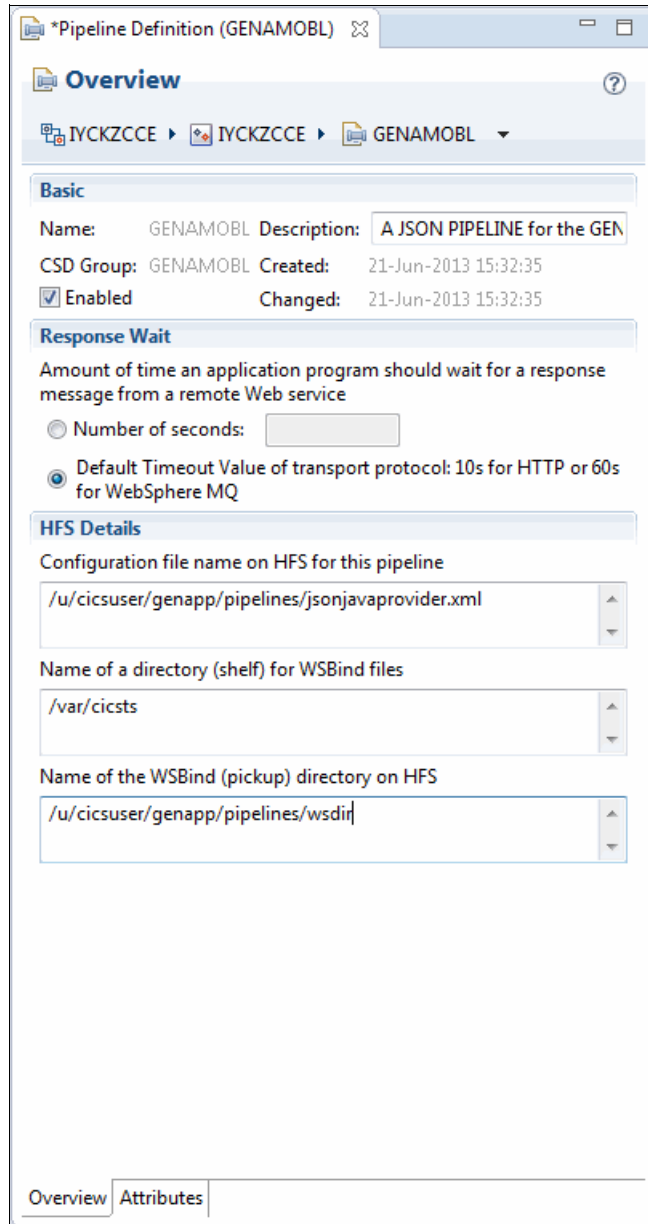


Figure 5-14 Editing a pipeline definition

6. Click **File** → **Save**.
7. Next, right-click the newly created PIPELINE in the Pipeline Definitions view and click **Install**, as shown in Figure 5-15 on page 45.

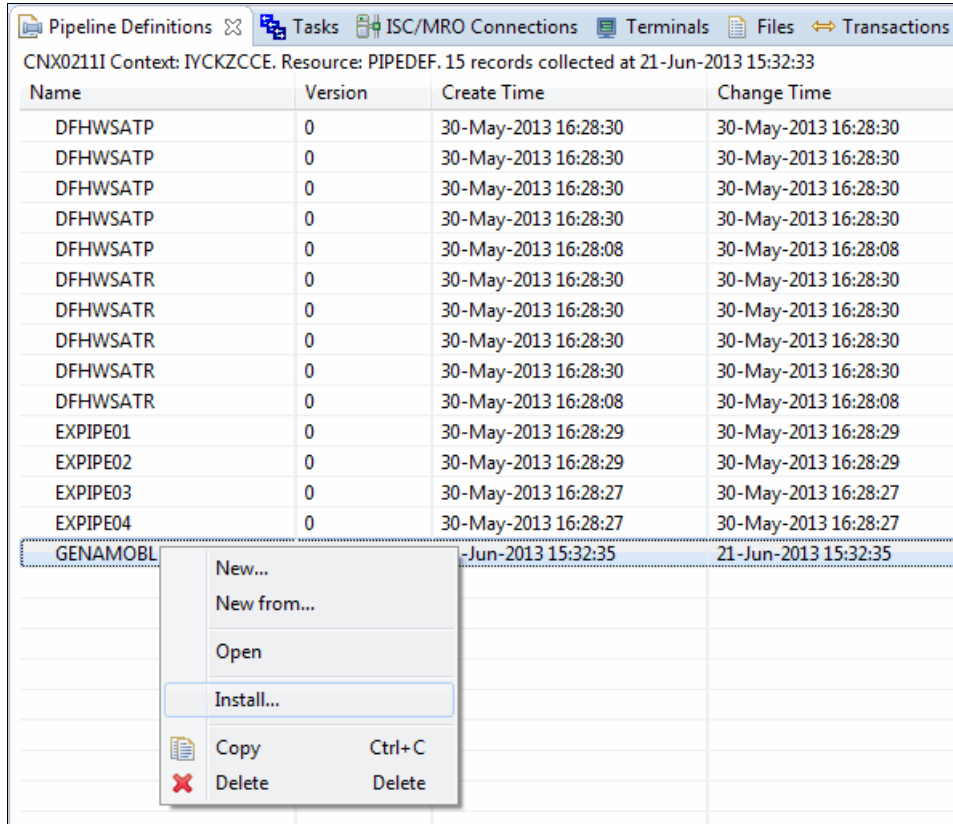


Figure 5-15 Installing a pipeline

8. When the Perform Operation window opens, select your CICS system and click **OK**.
If the operation failed, diagnose and correct the problem, then repeat the install operation.
9. Open the Pipelines view by clicking **Operations** → **Pipelines**. You will see your installed PIPELINE, as shown in Figure 5-16.

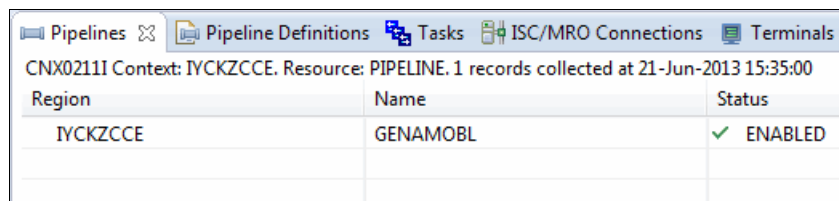


Figure 5-16 View of installed pipelines

As part of the installation of the PIPELINE, CICS will have created WEBSERVICE resources for each of the WSBIND files in the pickup directory. In this scenario, the directory was empty, so no web services were created. This is indicated by the messages issued to the CICS message log, as shown in Example 5-1.

Example 5-1 The messages issued when you install a PIPELINE resource

```
DFHRD0124 I 21/06/2013 15:34:49 IYCKZCCE CICSUSER CWWU INSTALL
PIPELINE(GENAMOBL)
DFHPI0703 I 21/06/2013 15:34:50 IYCKZCCE CICSUSER PIPELINE GENAMOBL is about to
scan the WSDIR directory.
```

```
DFHPI0704 I 21/06/2013 15:34:50 IYCKZCCE CICSUSER PIPELINE GENAMOBL Implicit
scan has completed. Number of wsbind files found in the WSDIR directory:
000000. Number of successful WEBSERVICE creates: 000000. Number of failed
WEBSERVICE creates: 000000.
```

10. After the WSBIND files are put into the pickup directory, perform a PIPELINE scan. This will perform an explicit scan of the directory and create WEBSERVICE and URIMAP resources for the WSBIND files.

Your CICS system is now ready for the scenarios described in Chapter 9, “Language structure to JSON schema scenario” on page 93, and Chapter 10, “JSON schema to language structure scenarios” on page 105.

5.3 CICS as a JSON client

In Chapter 11, “Developing a simple JSON web service client application” on page 127, we present a scenario where CICS acts as a client for a JSON web service. The example application performs a transformation between JSON and application data at run time. This CICS functionality is performed within a JVM server environment.

Therefore, if you want to follow this scenario, ensure you install a JVMSERVER with the JAVA_PIPELINE=YES option specified. The JVM server installed in the previous section is sufficient.

Later, more CICS resources will be installed using CICS Explorer as part of the scenario.



IBM Worklight configuration

This chapter provides information about the installation and configuration steps for both the IBM Worklight Studio and IBM Worklight Server. Worklight Studio is the Eclipse interface that a developer uses to implement a Worklight native or hybrid mobile application, and can be installed into an Eclipse instance. Worklight Server is where components developed for the server side (written in Worklight Studio), such as *adapters* and *custom server-side authentication logic*, run.

The following topics are covered in this chapter:

- ▶ “Worklight Studio” on page 48
- ▶ “Worklight Server” on page 52

6.1 Worklight Studio

IBM offers a Worklight Developer Edition enabling a programmer to get started trying out Worklight quickly and without initial cost. To download it, go to the following website:

<http://www.ibm.com/developerworks/mobile/worklight/index.html>

By signing in with your IBM ID (or signing up for one at no cost), you will be given an update site Uniform Resource Locator (URL) for an existing Eclipse installation. You must use Eclipse version 4.2.2 or later for Worklight Version 6.

If you do not already have an existing Eclipse installation, download Eclipse Juno 4.2.2 (or either Eclipse integrated development environment (IDE) for Java Platform, Enterprise Edition (Java EE) Developers, or Eclipse Classic) at the following website:

<http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/junosr2>

After installing the Eclipse option, use the installation steps for IBM Worklight available at the following website:

<http://www.ibm.com/developerworks/mobile/worklight/download/install.html>

After IBM Worklight is installed, perform the following steps to start a project:

1. Right-click in the **Project Explorer** and make a new Worklight Project. Click **Select a wizard**, then click **Worklight Project**. See Figure 6-1.

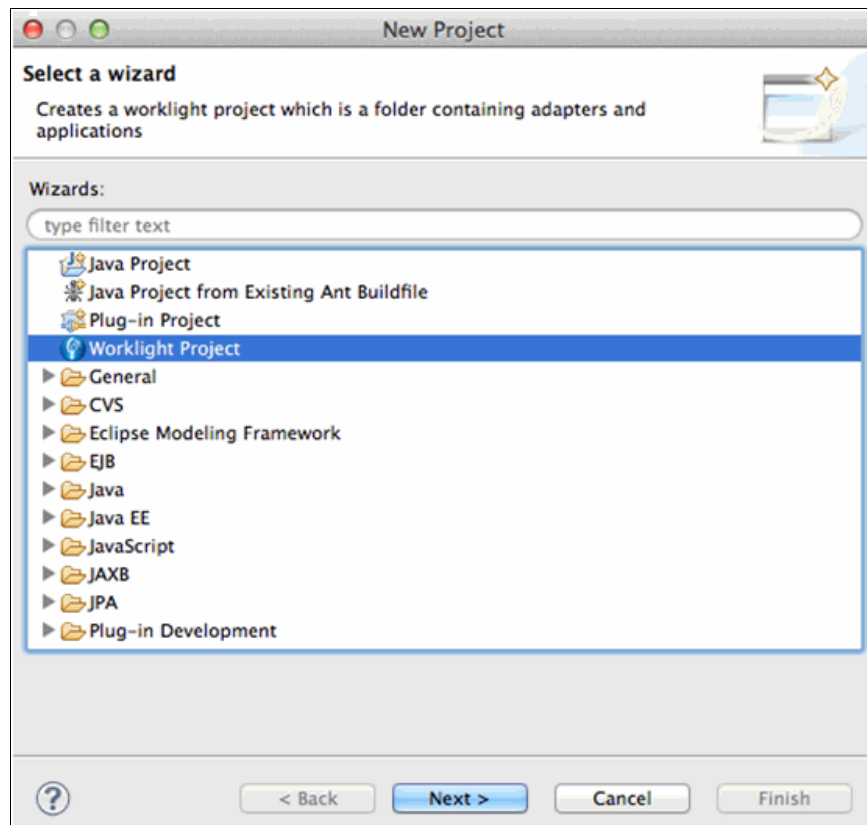


Figure 6-1 Create a new project

2. Give the application a name (for example, *CICS Test App*). Select **Hybrid Application**, as shown in Figure 6-2.

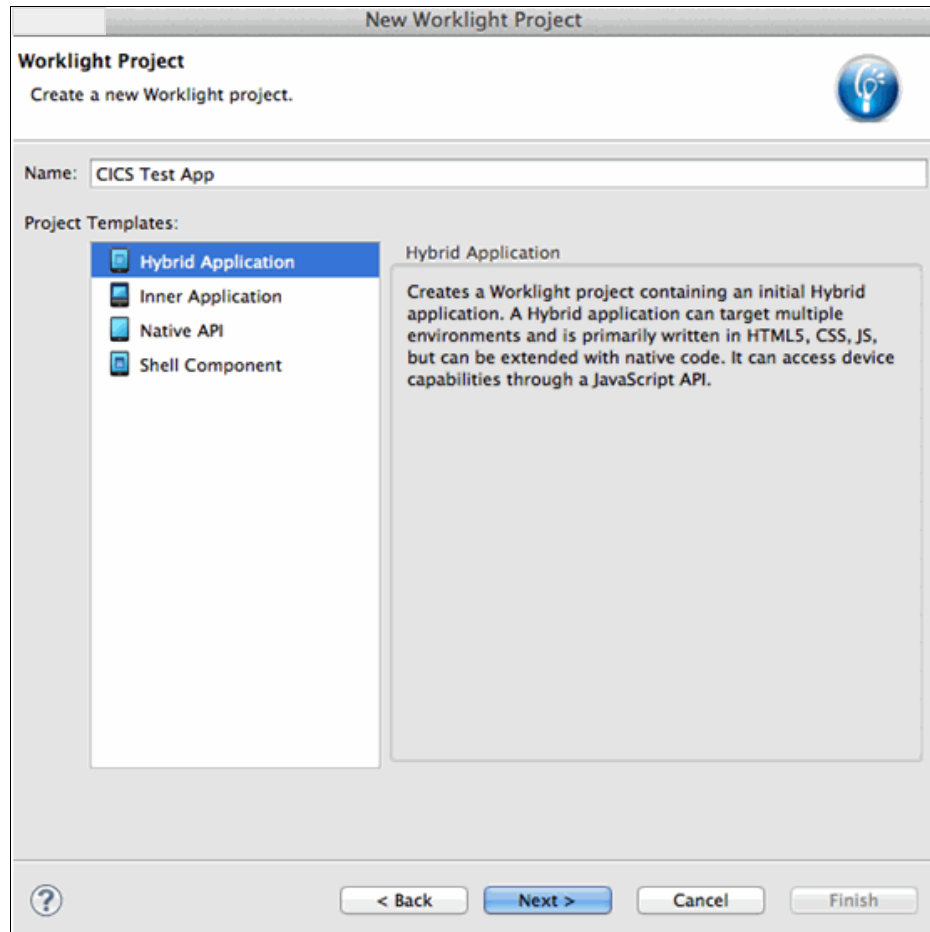


Figure 6-2 Worklight Project

3. You might add jQuery Mobile, Dojo Toolkit, or Sencha Touch for the user interface (UI) component of the application. For this example, **Dojo Toolkit** is selected, as shown in Figure 6-3.

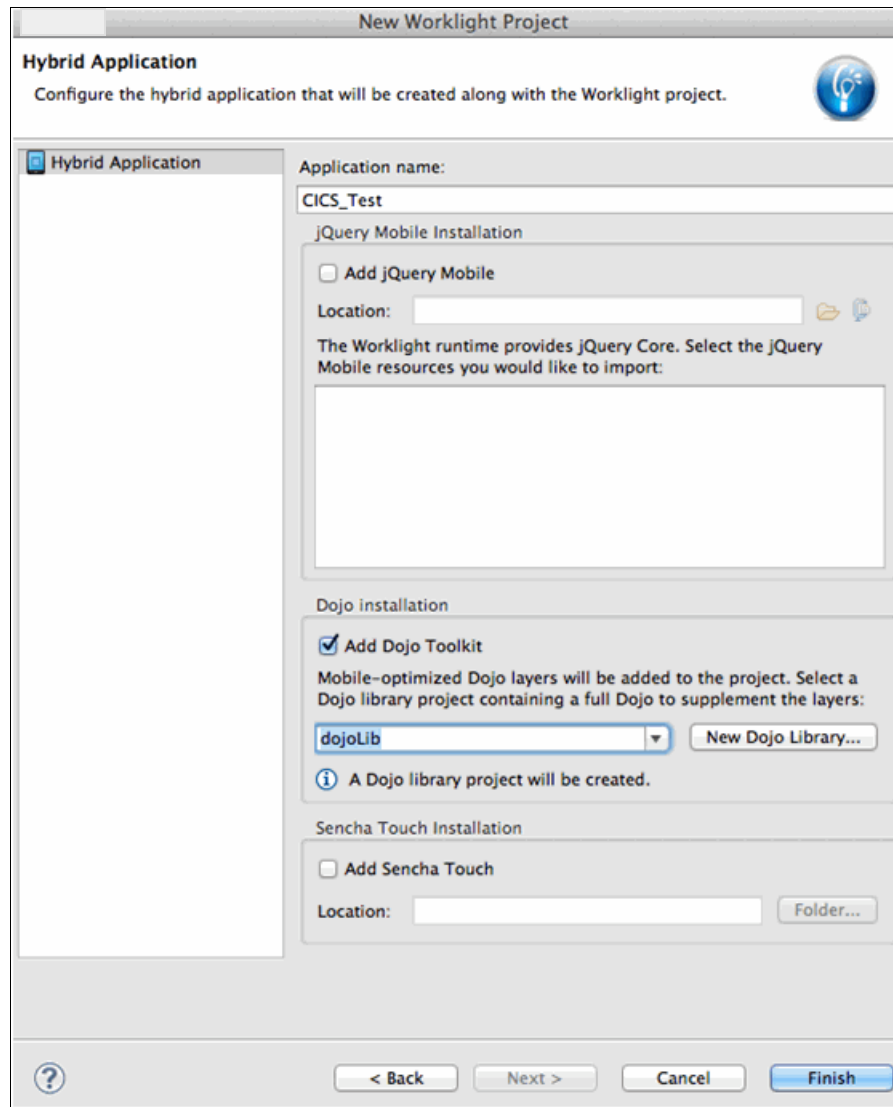


Figure 6-3 Hybrid Application

4. You will now notice the CICS Test App with the CICS_Test application in the newly created Worklight Project. See Figure 6-4.

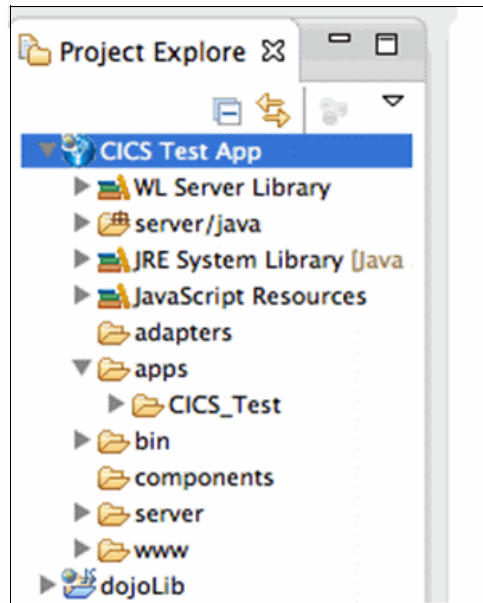


Figure 6-4 Run CICS Test App from the Project Explorer

- You can add mobile environments to the project. You need to add an environment to the application for each platform that you want the application to run under. Add a new environment by right-clicking the apps folder and select **New** → **Worklight Environment**. See Figure 6-5.

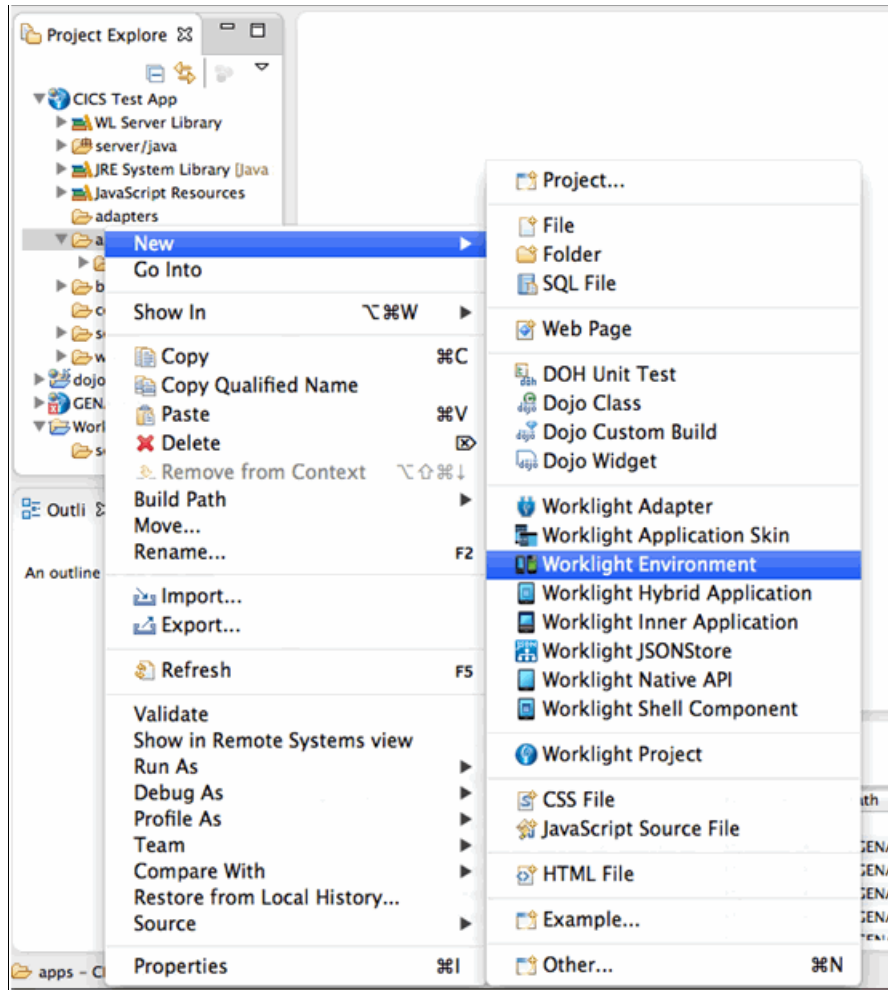


Figure 6-5 Run Worklight Environment from the Project Explorer

- You will need to have the appropriate software development kits (SDKs) for each platform set up on your machine to build for the different environments.

Notice a folder within the app for each environment. These folders are used for components that are specific to the environment that you are programming for. For example, you might want to have a special Cascading Style Sheets (CSS) layout for the iPad that is different than the iPhone.

6.2 Worklight Server

The Worklight Server is a dedicated server component for running Worklight adapters and custom authentication code written in Worklight Studio. The Worklight Console and application center all run from the Worklight Server, which acts as a management point for the Worklight installations.

Mobile clients are configured to connect to the Worklight Server through heartbeats. All data communication is managed to ensure that client updates are pushed to mobiles, services, and back-end applications running in environments such as Customer Information Control System (CICS).

The server is part of the Consumer or Enterprise edition when buying Worklight. Every Worklight Studio installation contains a Worklight Server for development and testing. In Worklight 6.0 and later, the server runs in Liberty, and in older versions of Worklight it runs in Jetty within Eclipse.

When installing Worklight Server you will need to make these decisions:

- ▶ Which application server you will use:
 - WebSphere Application Server Liberty Core
 - WebSphere Application Server
 - Apache Tomcat

This example uses the Liberty Core.

- ▶ Which database management system you will use:
 - IBM DB2
 - MySQL
 - Oracle
 - Apache Derby in embedded mode (included in the installation image)

This example uses IBM DB2.

To install the Worklight Server you will need to open IBM Installation Manager and add a new repository pointing to the location of the compressed file containing the Worklight Server installation files. If you are installing IBM DB2 for Worklight with the installation, you will also need to install this in a similar manner. IBM Installation Manager can be downloaded from the following website:

<http://www-01.ibm.com/support/docview.wss?uid=swg24033586>

To install the Worklight Server, use the following steps:

1. From the IBM Installation Manager, select the installation packages as shown in Figure 6-6. For this example, use IBM Worklight Server.

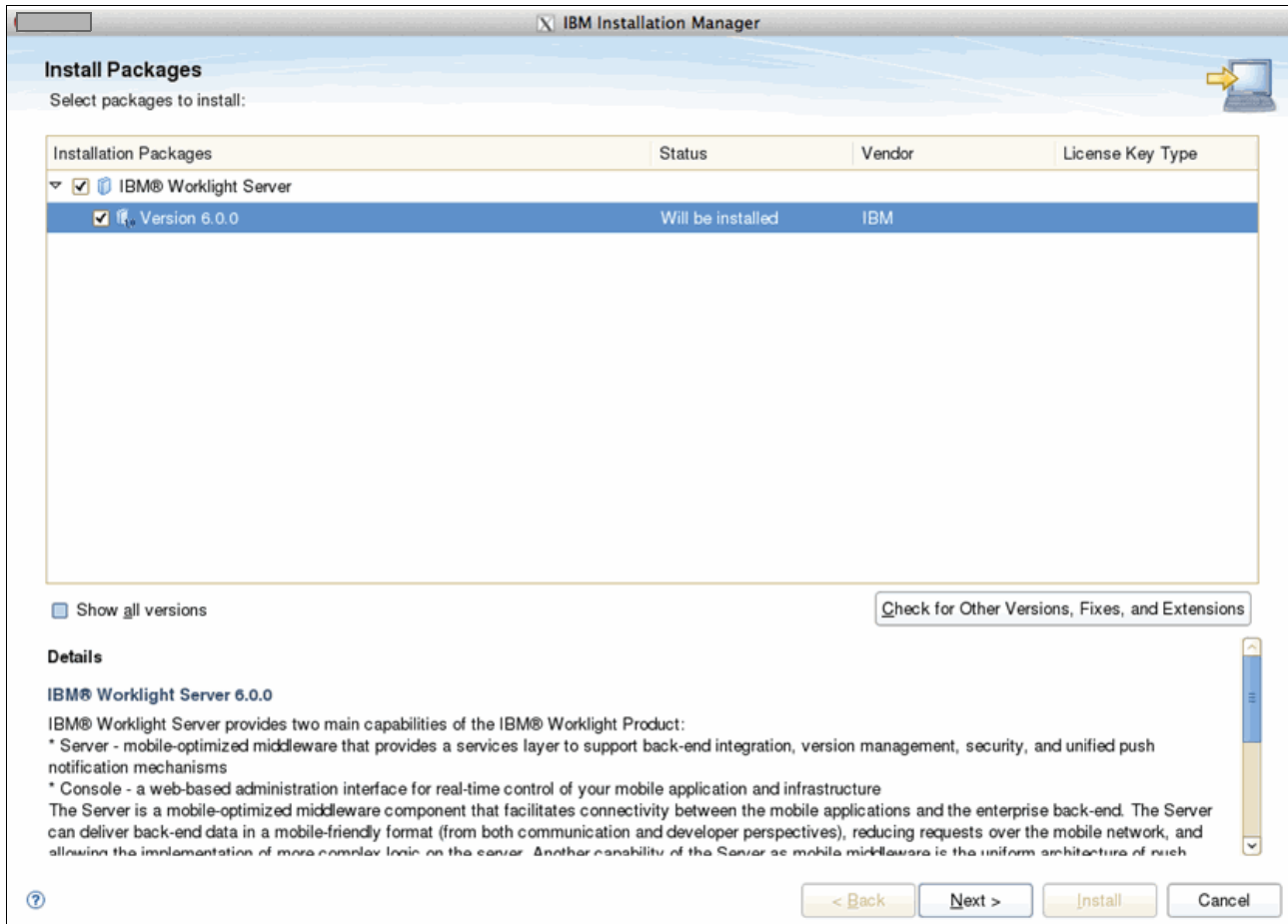


Figure 6-6 Select package to install

2. Accept the licensing agreements, as shown in Figure 6-7. For this example, use License Agreement Mobile Foundation Consumer Edition. Then, click **Next**.

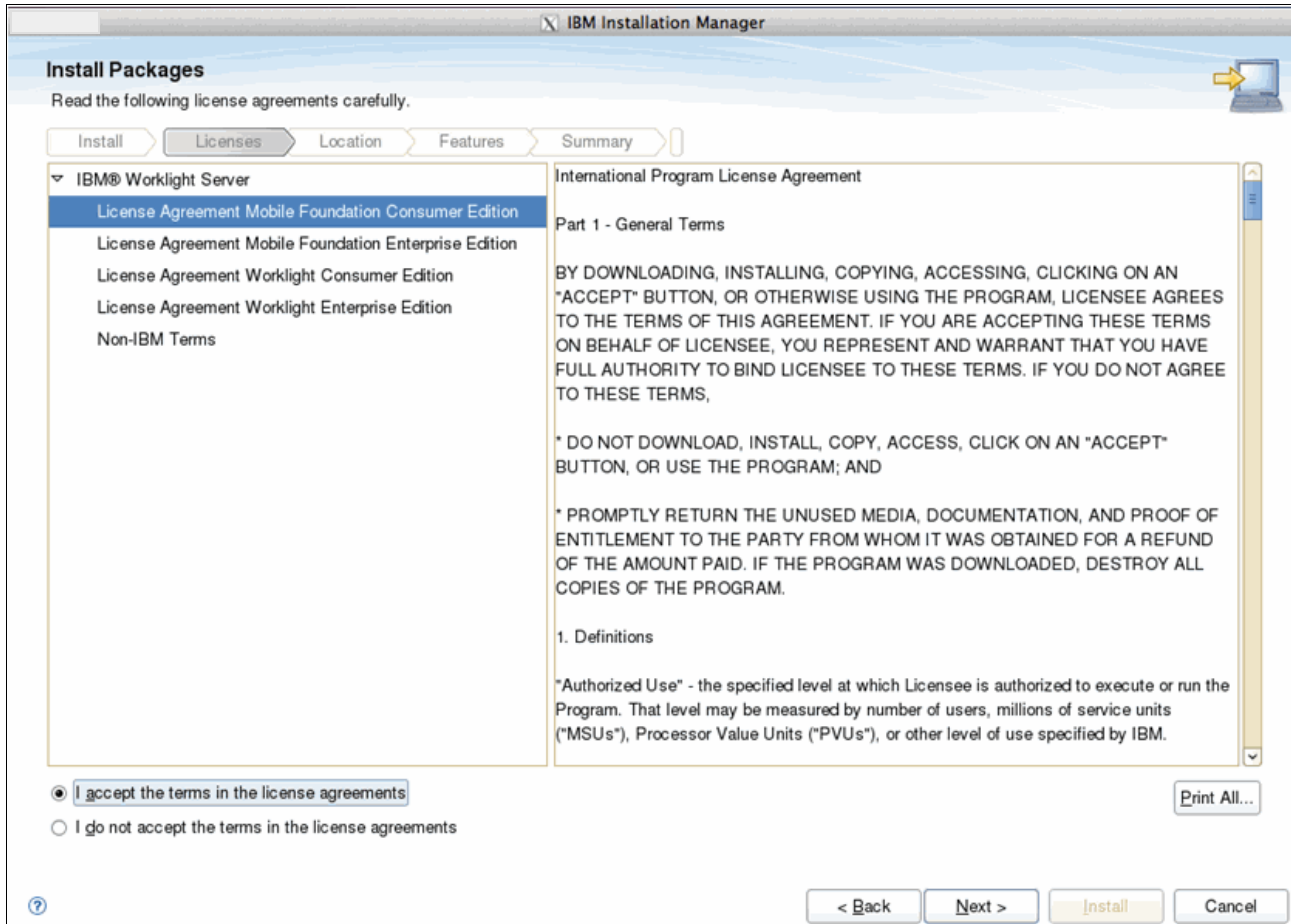


Figure 6-7 License agreements

3. Then choose the package group and installation location, as shown in Figure 6-8.

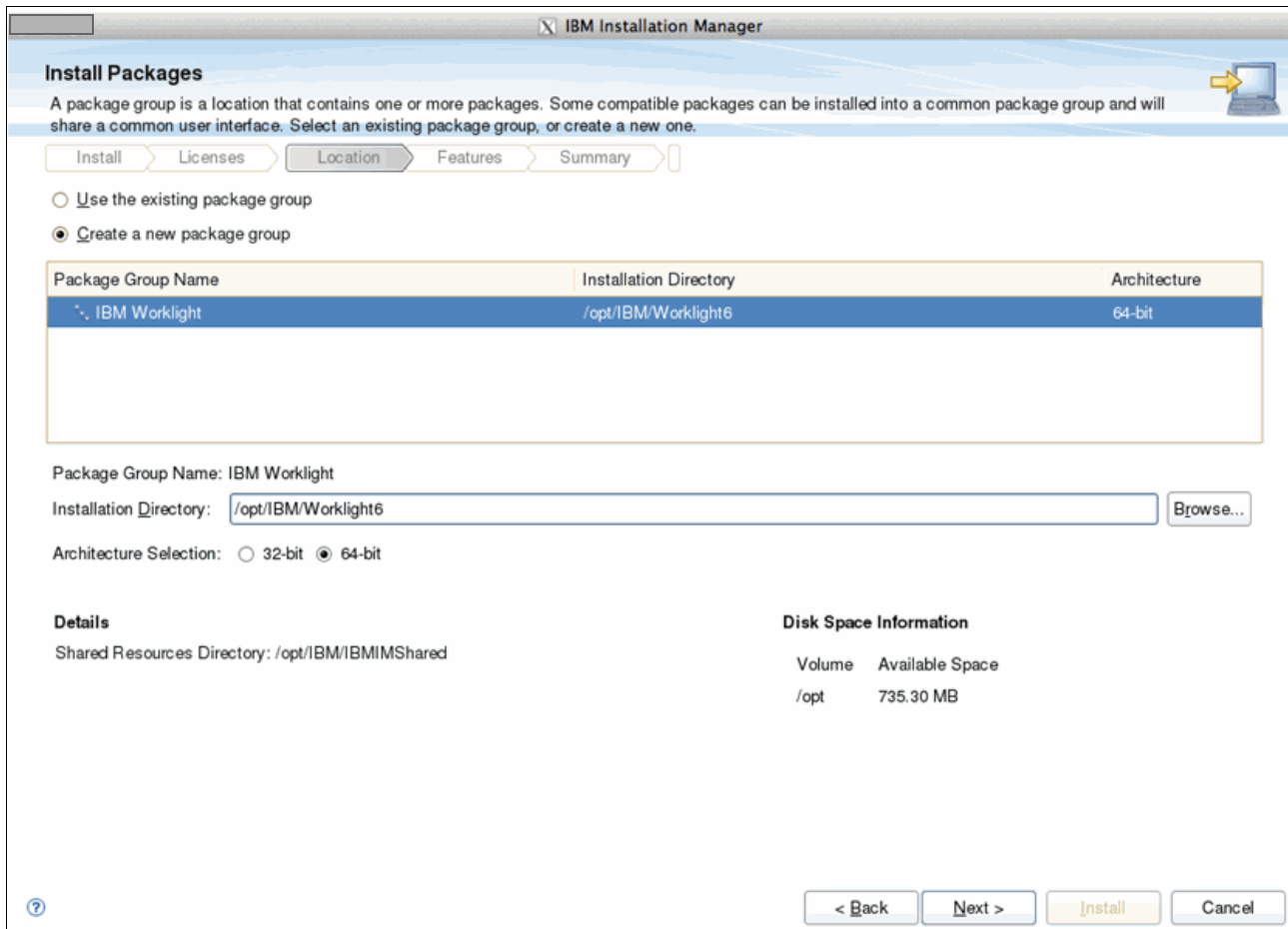


Figure 6-8 Create a new package group

4. Select the features to be installed and click **Next**. See Figure 6-9.

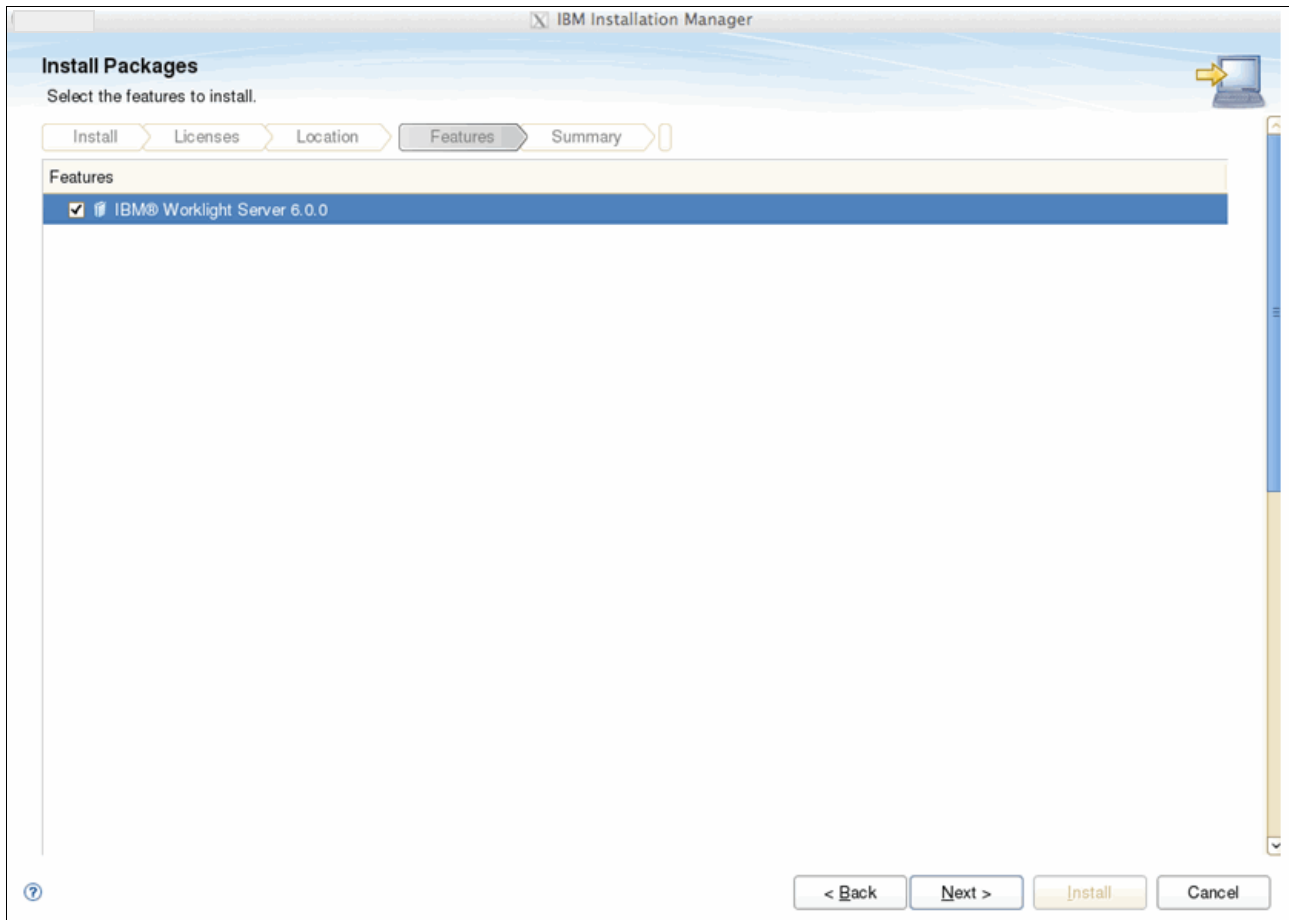


Figure 6-9 Select the features to install

5. Choose whether you want the application center to be installed. This acts as a private application center (such as the Android Google Play Store or iOS App Store) for your Worklight applications to be shared and installed on your enterprise's mobiles. See Figure 6-10.

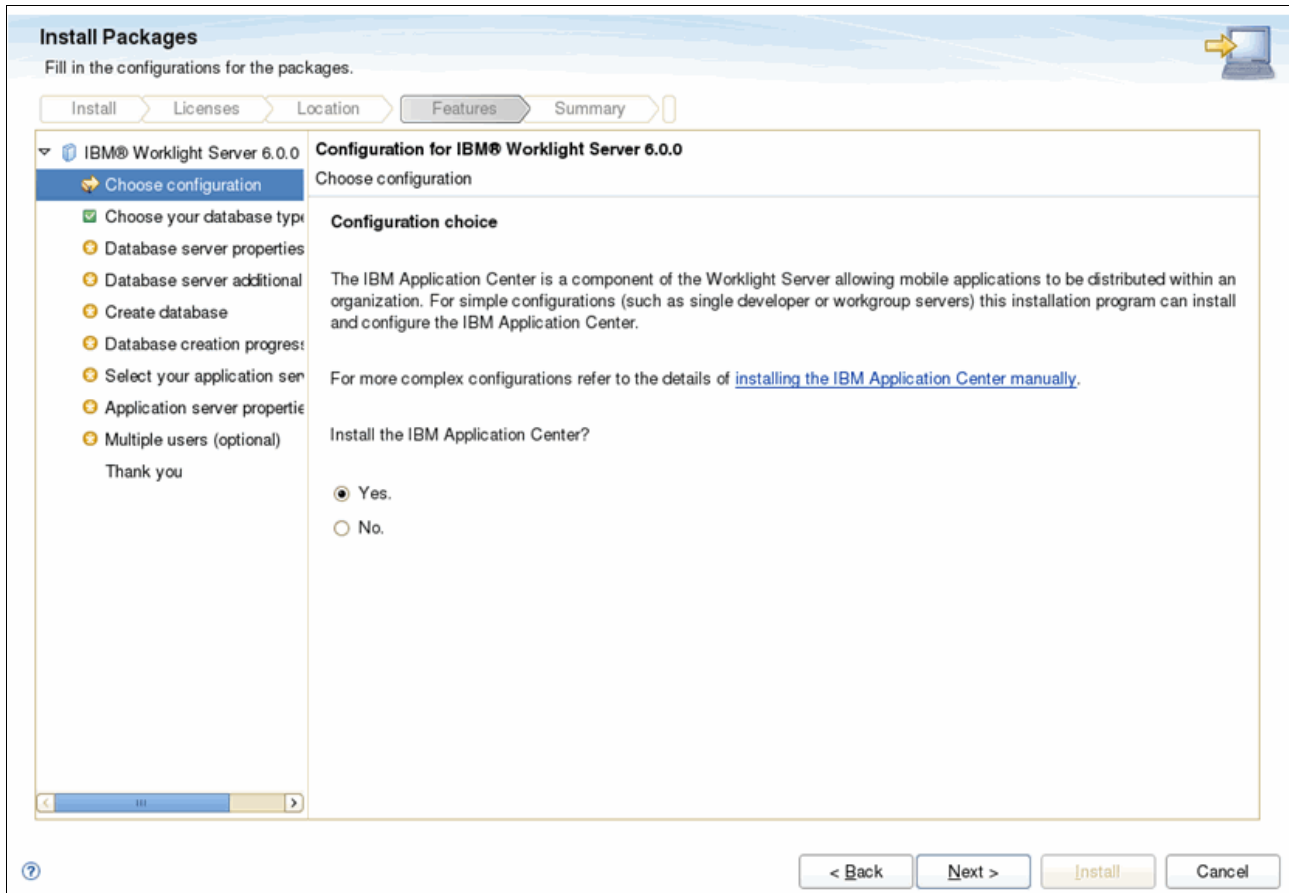


Figure 6-10 Choose configuration

6. Choose the database type. In this case, **DB2** is selected. See Figure 6-11.

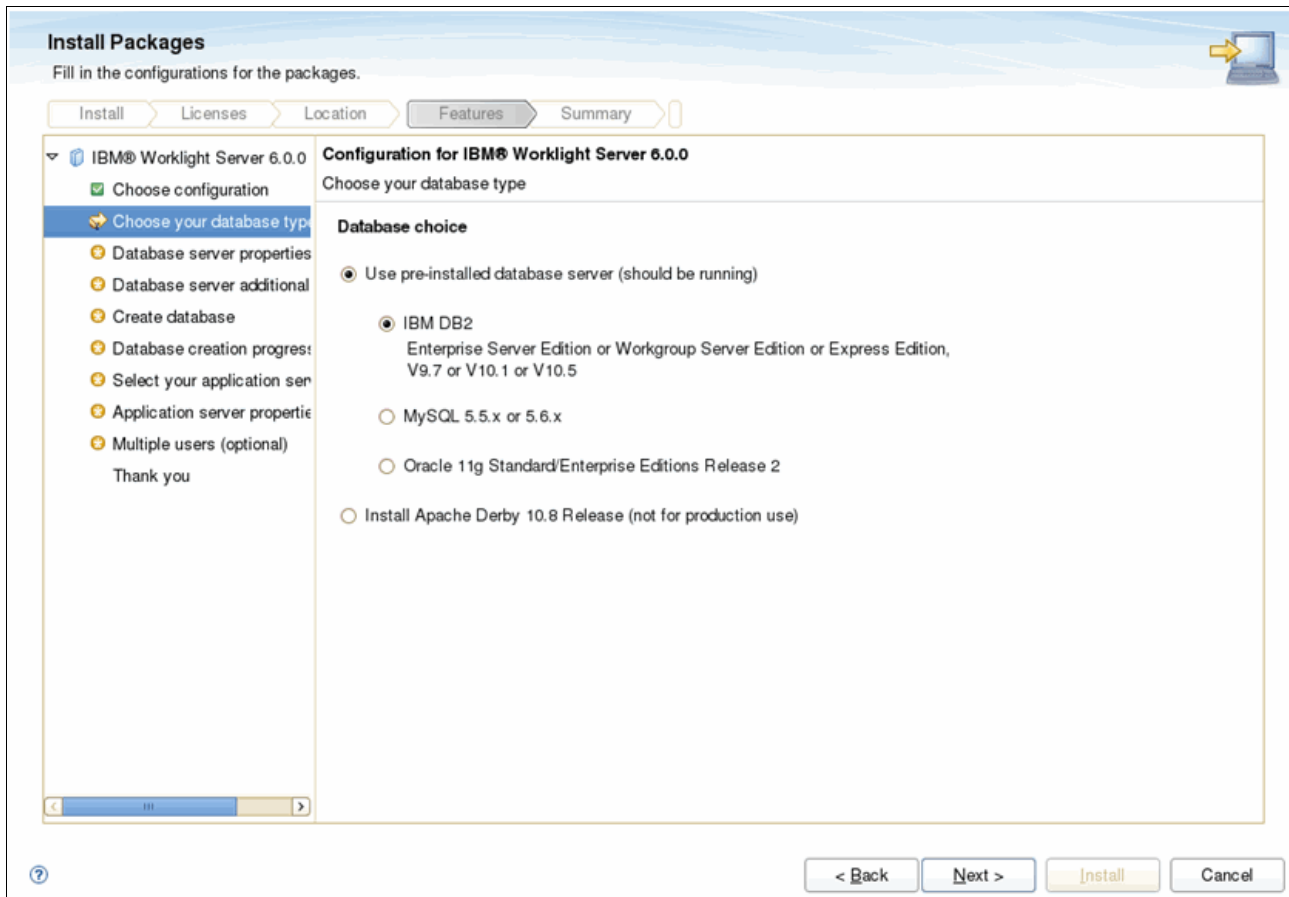


Figure 6-11 Choose your database type

7. Complete the database server properties. Choose the db2jcc4.jar file for DB2. In this case, the DB2 instance is running on the same host as the Worklight Server. See Figure 6-12.

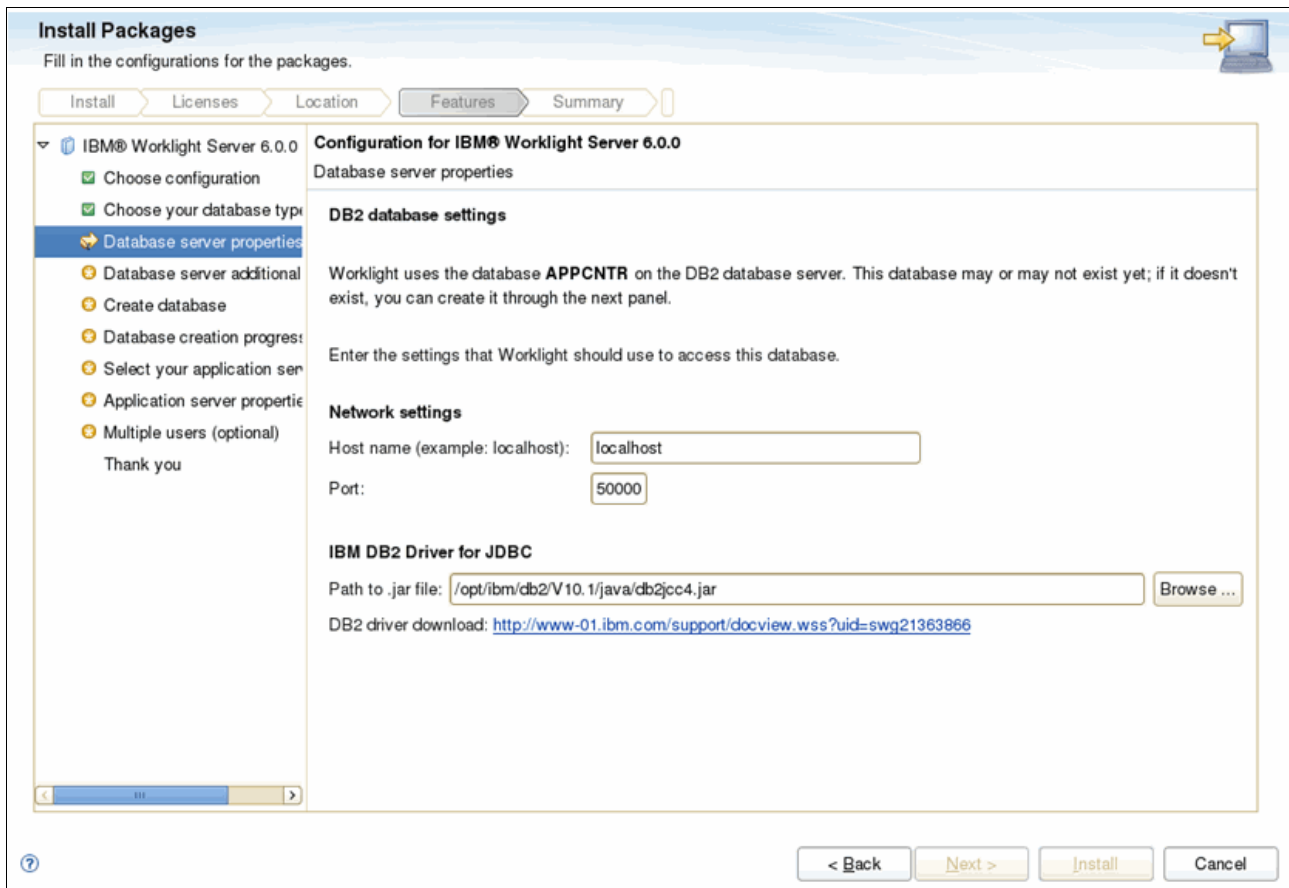


Figure 6-12 Install Packages

8. Enter the database connection settings, including user and password. In this case, the database is created manually using the DB2 command-line interface (CLI). See Figure 6-12 on page 60 and Figure 6-13.

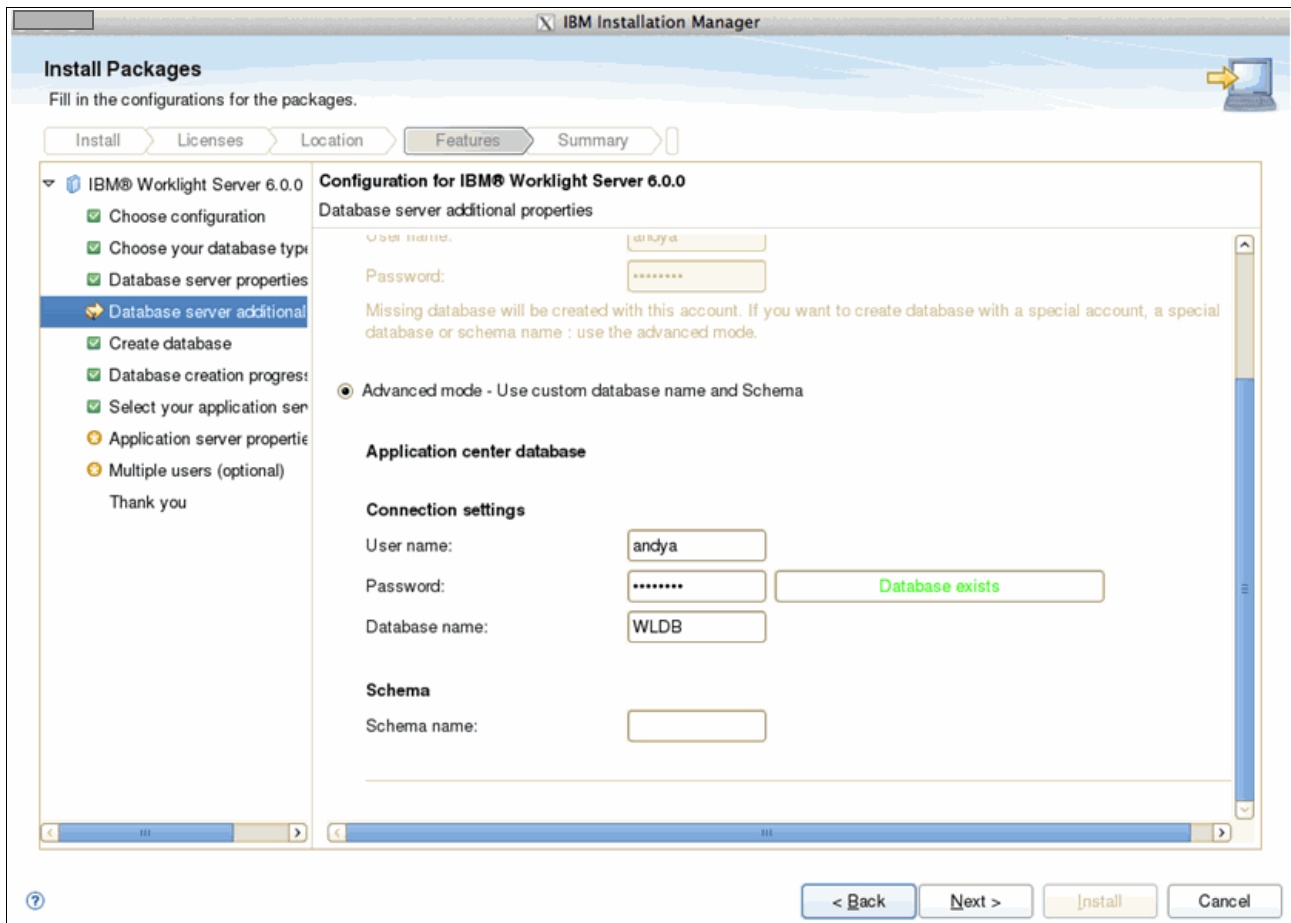


Figure 6-13 Database server additional properties

9. Installation Manager creates the database or confirms that it already exists. See Figure 6-14.

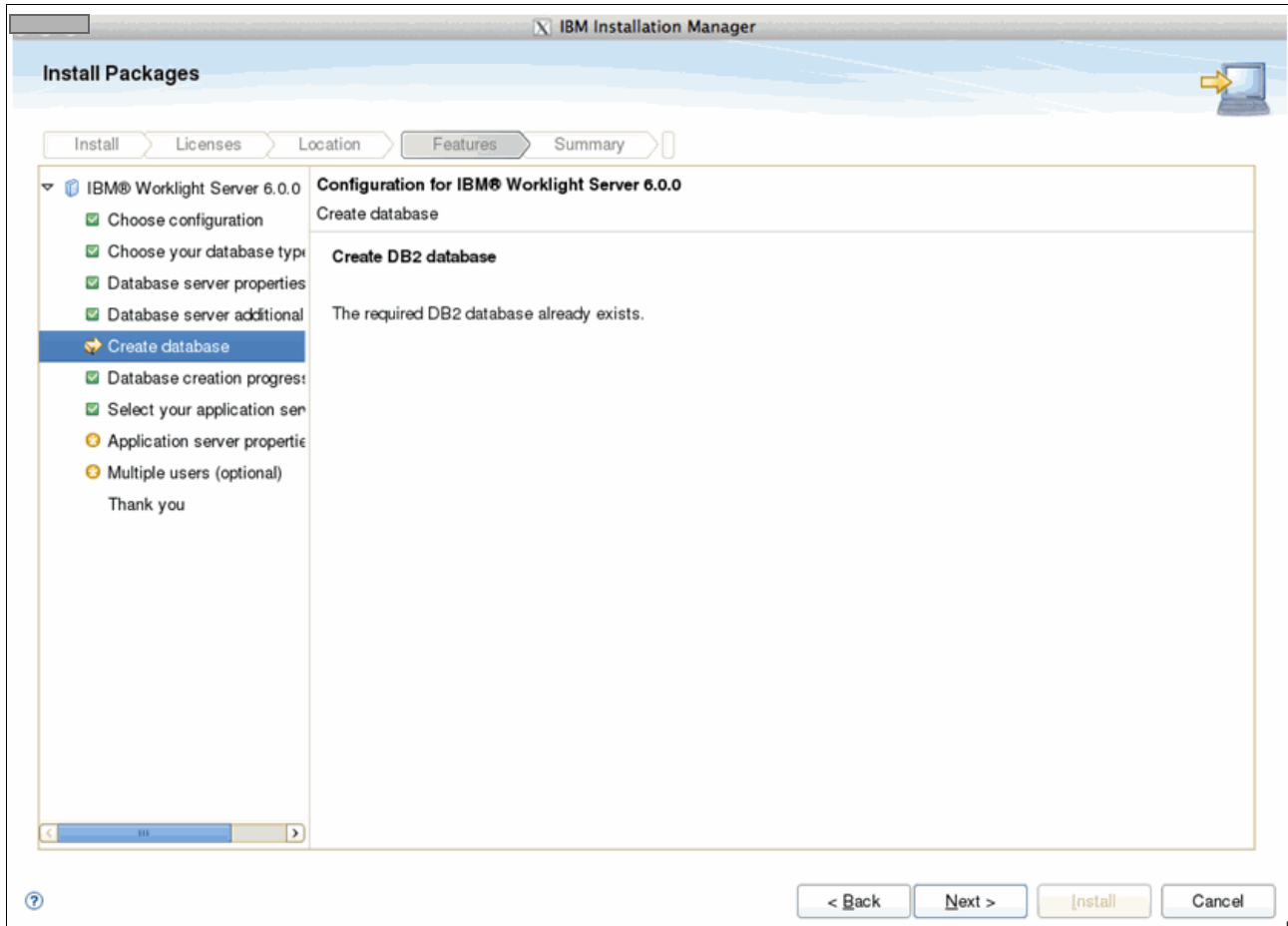


Figure 6-14 Create database

10. Select the Application Server (this example uses The WebSphere Application Server Liberty Profile).

Note that the Liberty profile should be installed separately, using Installation Manager, before proceeding with the Worklight 6 server installation. See Figure 6-15.

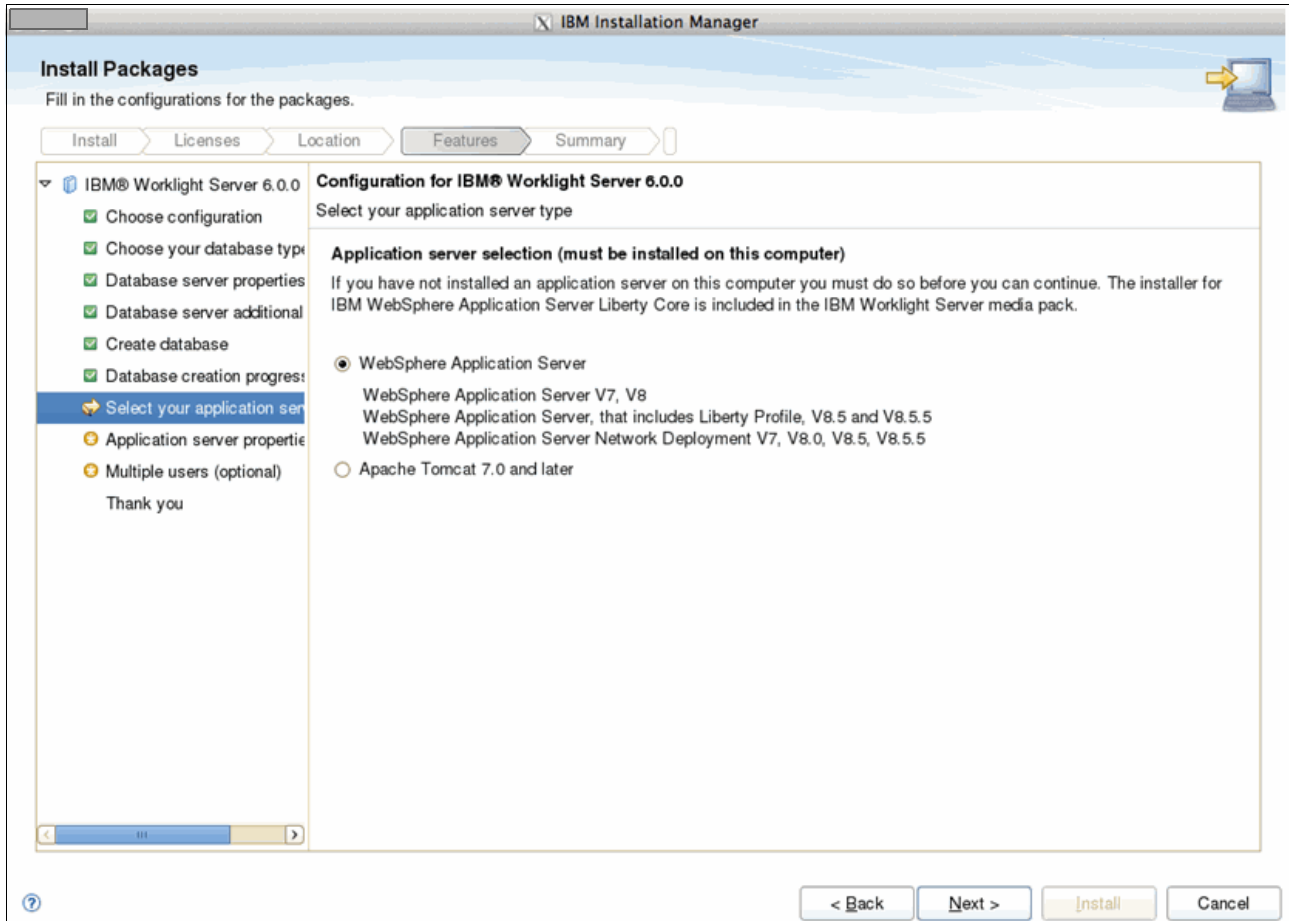


Figure 6-15 Select your application server type

11. Installation Manager checks that an application server was defined to the Liberty server configuration. If one cannot be detected, Installation Manager will inform the user. In this example, the defaultServer server was detected. This was created by going to the bin directory of the Liberty install and running the `./server create` command. See Figure 6-16.

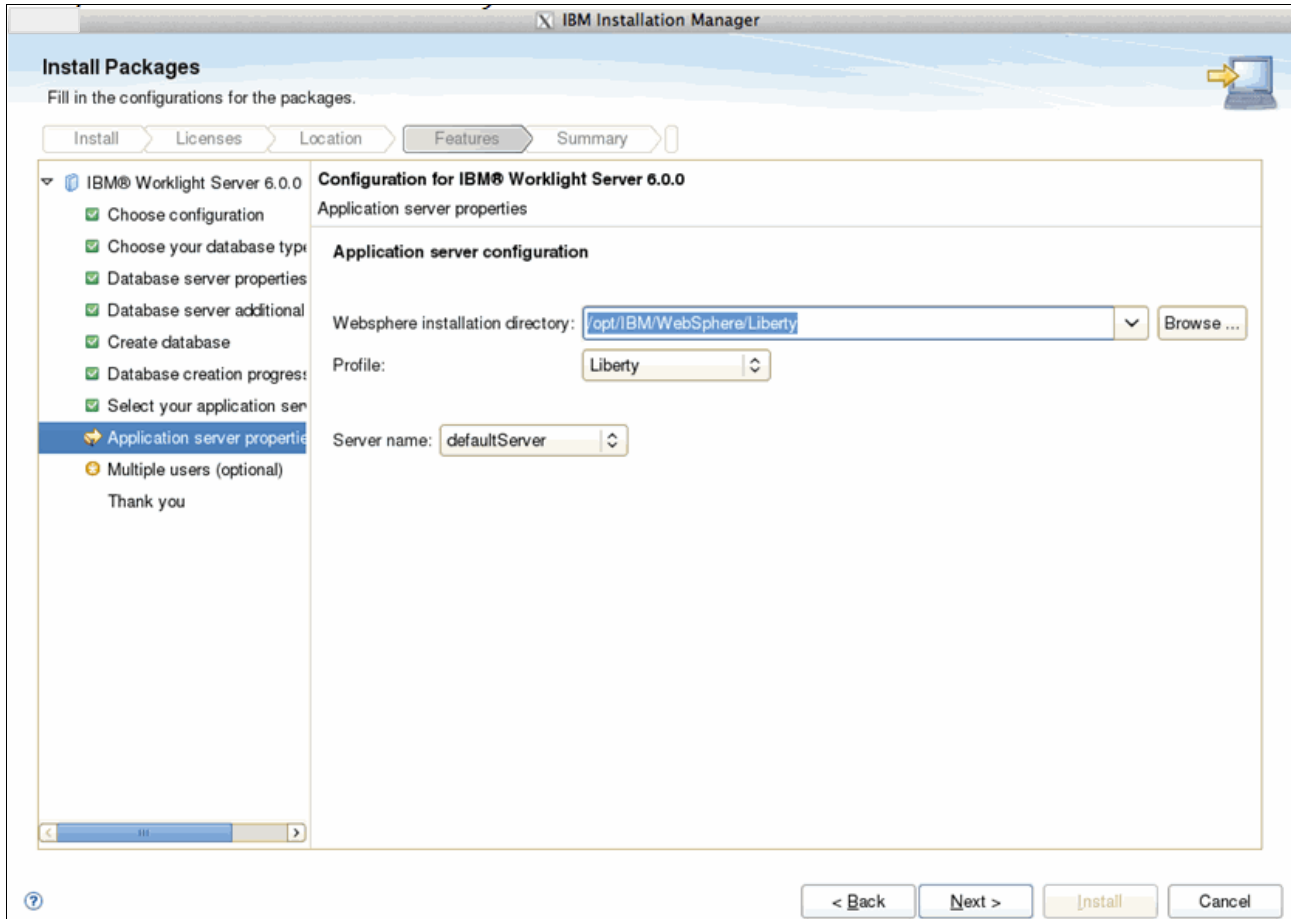


Figure 6-16 Application server properties

12. Select the configuration for the installation mode, either single or multiple users. Single user means only one specific user can start or stop the Worklight Server. Multiple users means all users of a specific group can configure the server. In this case, multiple users have been selected for the group adm. The installation manager highlights the users within the group for you to make it easier to see who will be able to control the server. See Figure 6-17.

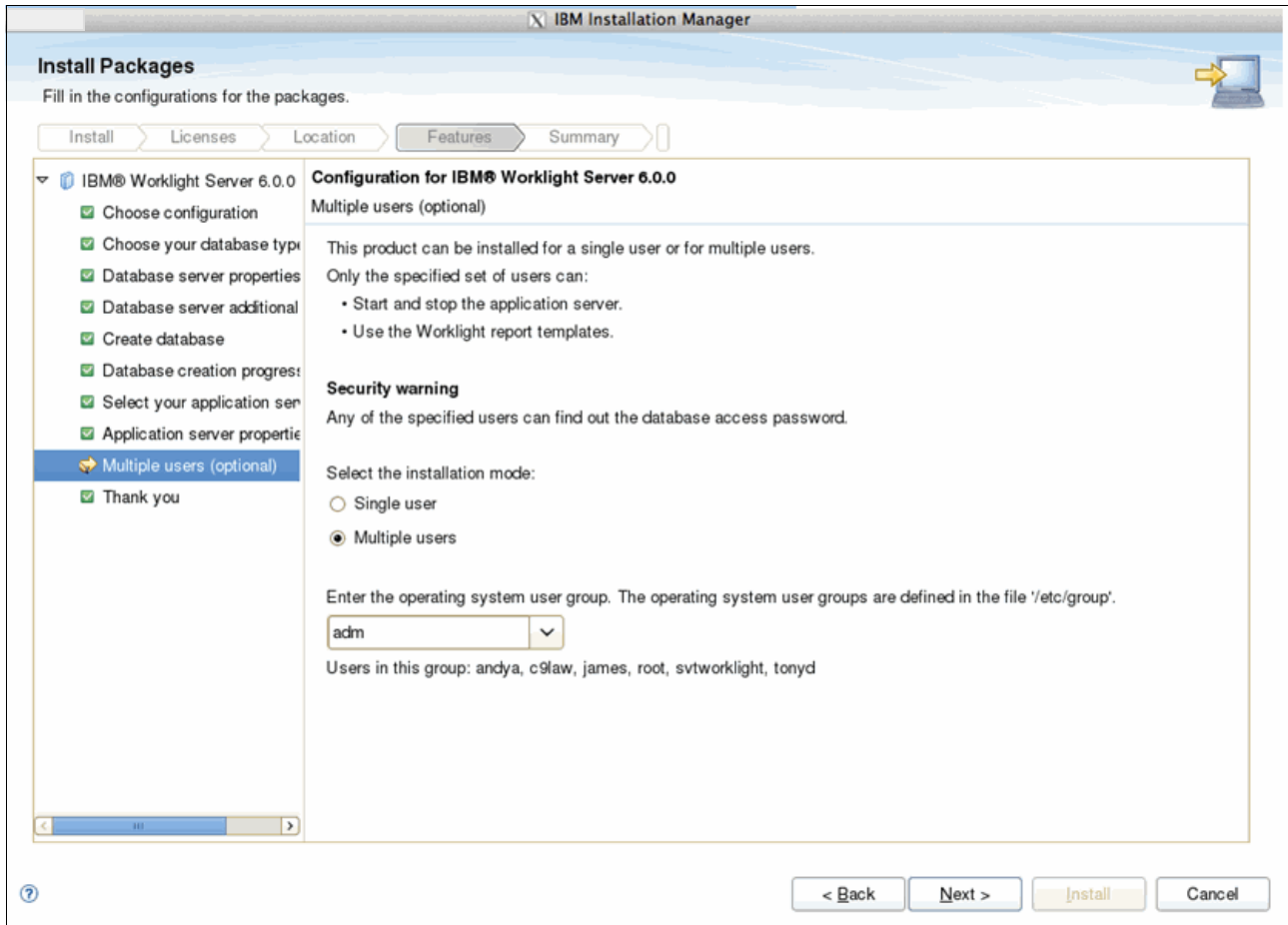


Figure 6-17 Multiple users (optional)

13. At this stage, the configuration is complete and the installation can begin. See Figure 6-18.

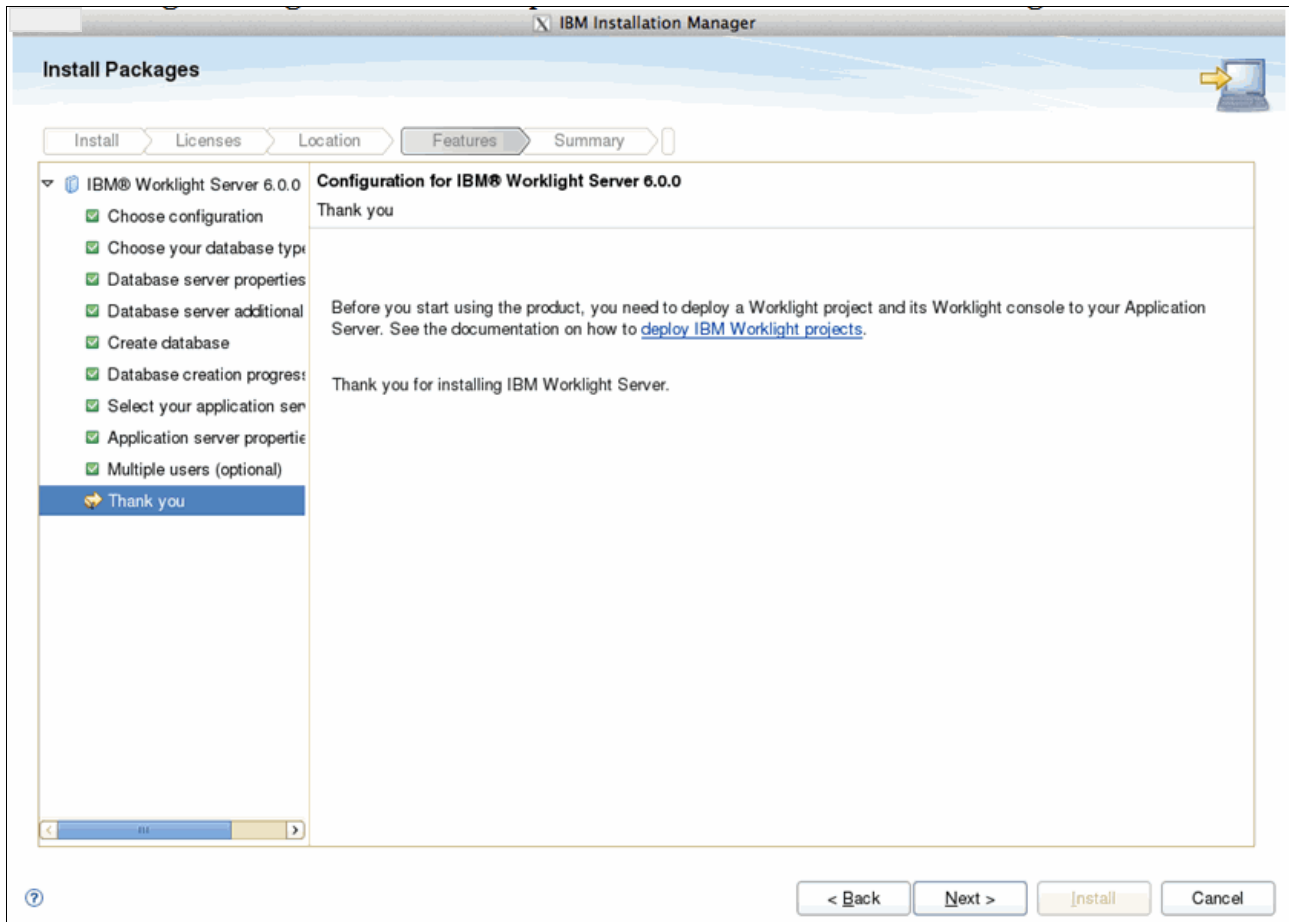


Figure 6-18 Configuration is complete and installation can begin

14. Confirm the summary details and click **Install**. See Figure 6-19.

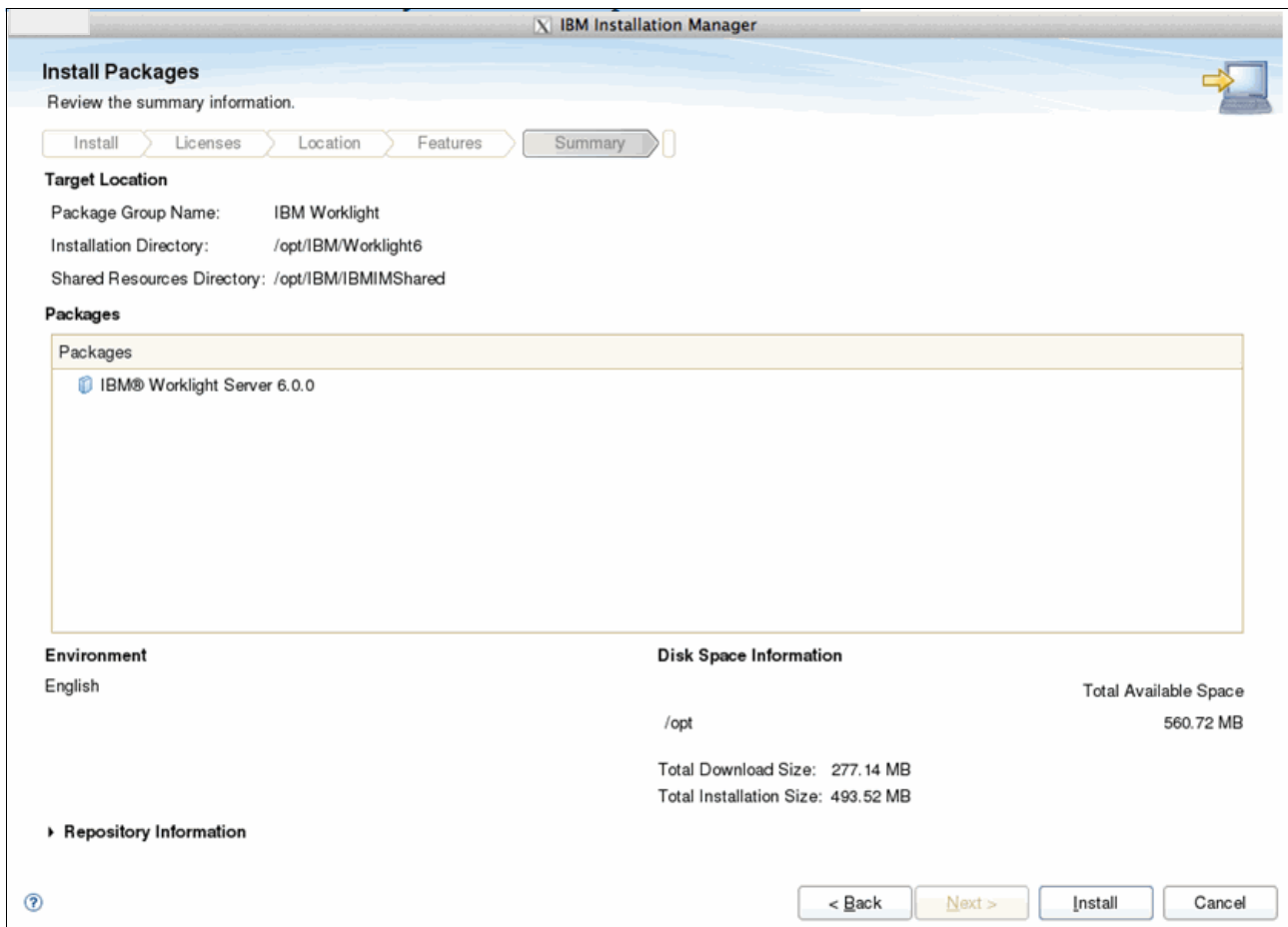


Figure 6-19 Summary window

15. A confirmation dialog that all installation is complete will be displayed, and the Installation Manager can now be closed. See Figure 6-20.

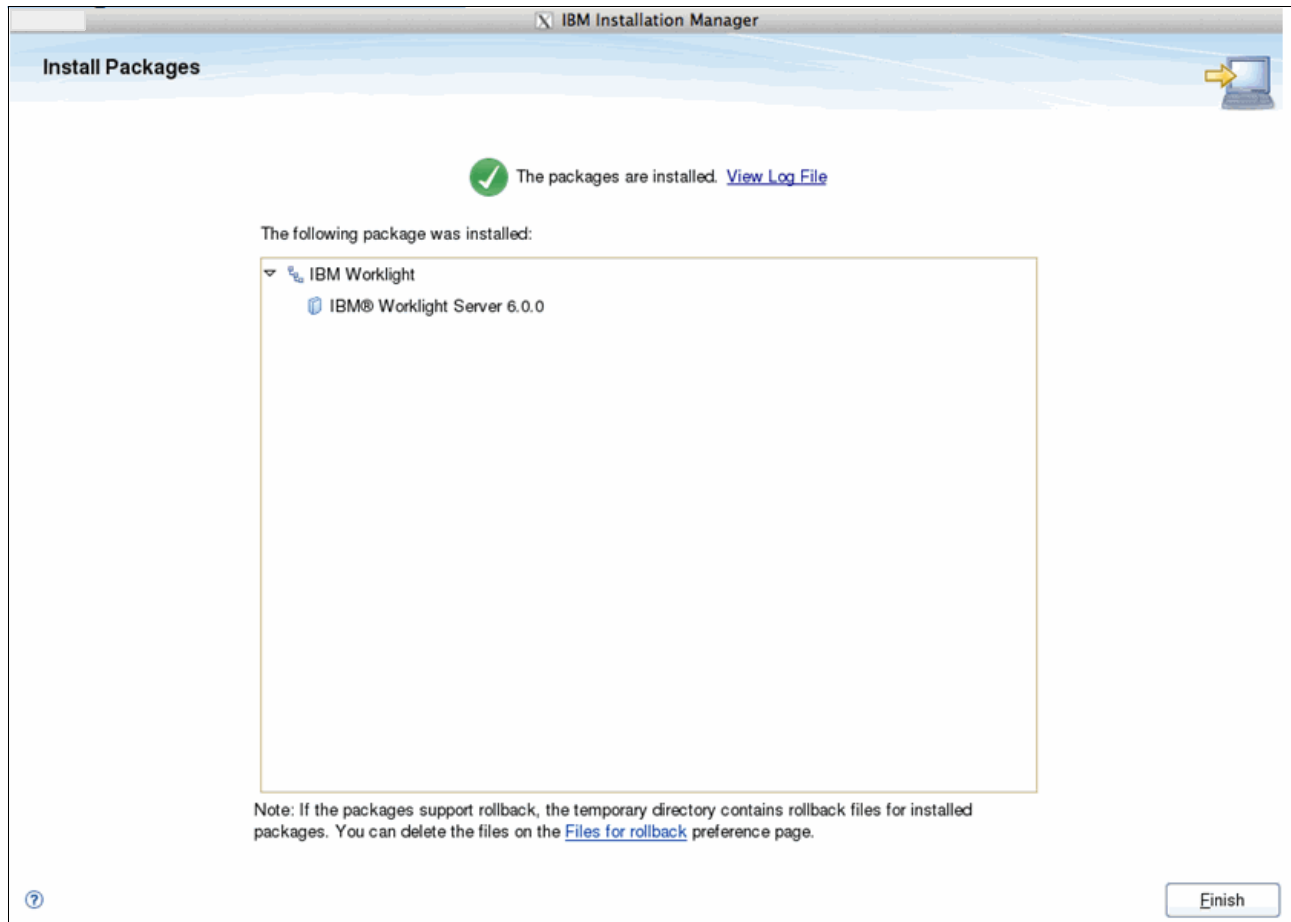


Figure 6-20 The packages are installed

16. Start the Worklight Server. Navigate to the application server directory. In this example, using WebSphere Liberty, it is in the following directory structure:

```
/opt/IBM/WebSphere/Liberty/bin and run ./<server> start <defaultServer>
```

In this case, *defaultServer* is the name of the Liberty server selected at step 12 on page 65.

17. The logs will highlight any problems present, so check the `messages.log` and `console.log` files under the following location:

```
/opt/IBM/WebSphere/Liberty/usr/servers/defaultServer/logs
```

18. Verify the login page. To do so, go to the following website:

```
http://<domain>:9080/appcenterconsole/login/login.html
```

A login page will open if the Worklight Application Center and Worklight have started successfully. The user name and password are both demo by default.



Security and workload management

Customer Information Control System (CICS) applications and their associated data constitute some of the most valuable assets owned by an enterprise. Therefore, the protection of these assets is an essential part of any CICS mobile project.

After a review of the main mobile security challenges, this chapter outlines the options for securing CICS JavaScript Object Notation (JSON) web services, reviews how products, such as Worklight and DataPower, can help, and then shows examples of security configurations in CICS and Worklight.

In this chapter, we provide information about the different techniques that can be used to provide high system availability and workload management for JSON web service applications. We summarize how high availability is provided across an IBM Parallel Sysplex®, and a multi-region approach for processing JSON web services workload.

This chapter contains the following topics:

- ▶ 7.1, “Security overview” on page 70
- ▶ 7.2, “Configuring security for JSON web services” on page 76
- ▶ 7.3, “Worklight security configuration” on page 78

7.1 Security overview

Every day, countless confidential transactions with financial institutions, online merchants, airlines, and various other retailers are performed on mobile devices. The biggest challenge that mobile service providers have is how to secure these services.

In addition to the normal web security challenges (authentication, authorization, confidentiality, data integrity, and nonrepudiation) mobile applications pose new challenges. There can be little or no control over the device, including when it is used, where it is used, who is using it, and for what it is used.

The following list notes some specific mobile security challenges:

- ▶ Who has access to the devices used to run your mobile applications?
- ▶ Are all of the devices that run your organization's mobile applications owned by your organization?
- ▶ What form of user authentication was implemented as part of the application? Has two-factor authentication been implemented?
- ▶ What sensitive information is stored or cached by mobile applications on the devices? Is this information encrypted? Could this data be deleted remotely?
- ▶ Can the mobile devices used to run your mobile applications be remotely disabled?
- ▶ Is communication between mobile devices and your enterprise network and systems secure?
- ▶ Is the distribution of your mobile applications controlled appropriately?

Addressing all of these challenges is clearly beyond the scope of CICS. However, securing access to resources managed by CICS is a significant part of the challenge. The chapter focuses on this area. It takes a brief look at the role of other products. For example, this chapter provides information about how Worklight server provides mobile application security, how DataPower can act as a mobile gateway, and how IBM Endpoint Manager for Mobile Devices addresses mobile device management.

7.1.1 Security principals and concepts

Mobile security is achieved through compliance with the following security principles:

Authentication	Ensures that the identities of both the sender and receiver of the mobile transaction are true.
Authorization	Grants a mobile user, system, or process either complete or restricted access to a resource.
Confidentiality	Protects sensitive data from unauthorized disclosure.
Integrity	Ensures that information that arrives at a destination is untampered.
Nonrepudiation	Proves that a mobile transaction occurred, or that a message was sent or received.

Consider the risks if inadequate authentication and authorization mechanisms are put in place. Thieves of stolen devices might be able to retrieve user credentials from the mobile device, or cyber criminals might bypass authentication controls. To address these challenges, multi-factor authentication is normally required (for example, verification of the device, user, *and* mobile application).

Consider the consequences if inadequate confidentiality, integrity, and nonrepudiation mechanisms are put in place. Mobile users' confidential information, such as bank account details, can be lost, cyber criminals might be able to modify the amounts of money being transferred, and mobile users might be able to deny the transactions that they performed. To address these challenges, it is normally required to use encryption, and also to ensure that new mobile security features are integrated into the existing enterprise security infrastructure.

7.1.2 CICS security options for JSON web services

In a CICS environment, the assets that you normally want to protect are the application programs and the resources that are accessed by the application programs. To prevent disclosure, destruction, or corruption of these assets, you must control access to the CICS region, and to different CICS components.

You can limit the activities of a CICS user to only those functions that the user is authorized to use by implementing one or more of the CICS security mechanisms that protect transactions, resources, and commands.

When CICS security is active, requests to attach transactions, and requests by transactions to access resources, are associated with a user ID. When a user makes such a request, CICS calls the external security manager, such as IBM Resource Access Control Facility (RACF®), to determine if the user ID has the authority to complete the request. If the user ID does not have the correct authority, CICS denies the request.

In some cases, a user is a human operator, interacting with CICS through a terminal or a workstation. In this case, the security scenario is straightforward, in that any transactions started by the signed-on user will automatically be authorized against the appropriate user ID.

However, in the case of a mobile user using a web service client application, it is unlikely that the mobile user will have a RACF user ID. Therefore, you need to consider how the user will be authenticated to CICS, and what user ID will be associated with the CICS transaction. You also need to consider how the confidentiality and integrity of the message will be protected.

Transport security

For Hypertext Transfer Protocol (HTTP) connections from mobile devices, there are two ways that the mobile user can be authenticated using transport security:

- ▶ An HTTP client can provide HTTP basic authentication information (a user ID and password). The CICS transaction that services the client's request, and further requests made by that transaction, are associated with that user ID.

Note: A likely issue with this approach is the question of what basic authentication credentials a mobile user would use. Basic authentication is also not considered especially secure unless combined with Transport Layer Security or Secure Sockets Layer (TLS/SSL).

- ▶ A client program that is communicating with CICS using TLS or SSL can supply a client certificate to identify itself. The security manager maps the certificate to a user ID. The transaction that services the client's request, and further requests made by that transaction, are associated with that user ID.

Note: A likely issue with this approach is that TLS/SSL can use many compute resources and might not be appropriate for a mobile device.

CICS user IDs

Figure 7-1 shows a security scenario in which a mobile application sends a JSON web service request to CICS over an HTTP or HTTP over SSL (HTTPS) connection. The following different user IDs are shown:

- Flowed user ID** This is a user ID that is flowed with the request (either in the JSON data or in an HTTP header). CICS enables a custom handler in the pipeline to extract such a user ID and use it to set the DFHW-USERID container. The target application then runs in a new task that is associated with this user ID.
- Transport user ID** The transport-based user ID can be set using either basic authentication or SSL client authentication (see “Transport security” on page 71).
- URIMAP user ID** A Uniform Resource Identifier (URI) mapping (URIMAP) resource definition matches the URIs of web service requests. The URIMAP associates a URI for the request with a PIPELINE and WEBSERVICE resource that specifies the processing to be performed. You can use a URIMAP to specify the user ID under which the CICS task runs (known as the pipeline alias transaction).

You can also use the URIMAP to set the name of the transaction that CICS uses for running the CICS task (the default is CPIH).
- Default user ID** When a user does not sign on, CICS assigns a default user ID to the user. It is specified in the SIT parameter **DFLTUSER**. In the absence of more explicit identification, it is used to identify Transmission Control Protocol/Internet Protocol (TCP/IP) clients that connect to CICS. You should not give much authority to the default user ID.
- Region user ID** The CICS region user ID is used for authorization checking when the CICS system (rather than an individual user of the system) requests access to system resources, such as CICS data sets and other servers.

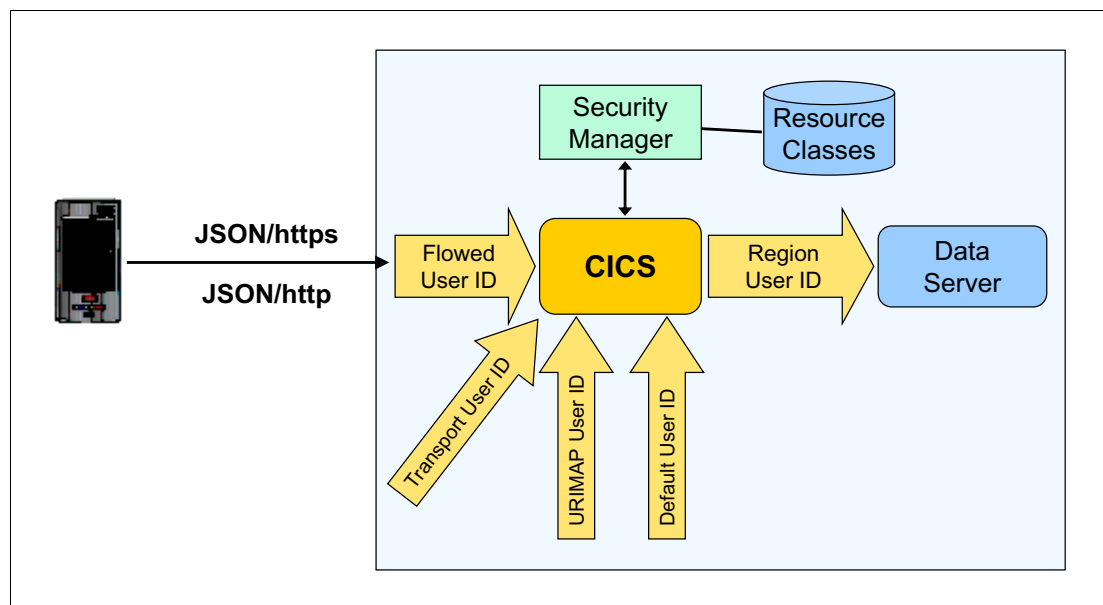


Figure 7-1 CICS mobile security scenario

It is possible that for a single JSON web service request, transported by HTTP, multiple methods for setting the user ID will be used at the same time. In this event, CICS uses the following order of precedence for determining the user ID associated with the CICS task:

1. A user ID inserted into the DFHW-USERID container by a message handler that is included in the service provider pipeline. This user ID might be extracted from a token in the header or body of the HTTP request.
2. A user ID obtained from the mobile client using basic authentication, or a user ID associated with a client certificate.
3. A user ID specified in the URIMAP definition for the request.
4. The CICS default user ID, if no other user ID can be determined.

7.1.3 CICS mobile security topologies

Previously, Figure 7-1 on page 72 showed a mobile device connecting directly to CICS. However, in most cases mobile devices will connect to another server before the request is passed on to CICS. Figure 7-2 shows two common CICS mobile topologies.

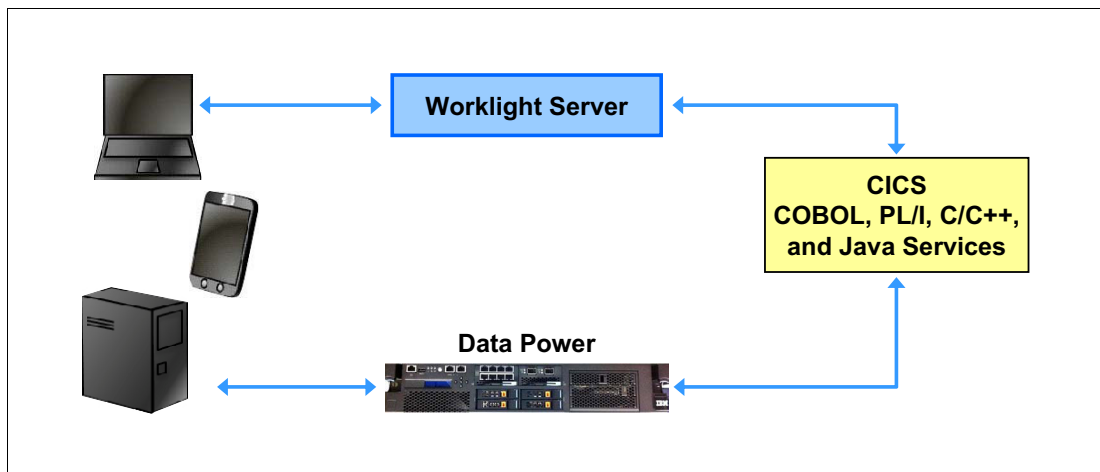


Figure 7-2 CICS mobile topologies

The following list notes several advantages of using an intermediary server:

- ▶ The intermediary server normally supports a wider range of mobile authentication mechanisms.
- ▶ The intermediary server can enable mobile single sign-on (SSO). SSO is an authentication process in which a user can access more than one system or application by entering a single credential (for example, a user ID and password).
- ▶ The intermediary server can protect CICS against unauthorized access and attacks.

Note: Worklight and DataPower can be used together to create a secure mobile infrastructure.

When an intermediary server is used to authenticate mobile users on behalf of CICS, it is important to establish a trust relationship between the intermediary server and CICS.

Transport security mechanisms, such as basic authentication and SSL client authentication, can be used to establish this trust relationship (see 7.3, “Worklight security configuration” on page 78 for examples of using transport-based security between Worklight server and CICS).

7.1.4 Worklight security

IBM Worklight provides a set of security capabilities that address a wide range of mobile security objectives, including the following list:

- ▶ Protecting data on the device.

It is common for the mobile application user to have access to sensitive data that can be stored on the mobile device. However, this data stored on-device can potentially be stolen or tampered with by malware existing on the device. In the event that the device is lost or stolen, this sensitive data can be extracted by unauthorized third parties.

In addition, the mobile application can be required to function in an offline context (without any back-end connectivity), and at the same time require that only authenticated users be given access to the data stored on the mobile device.

Worklight provides encrypted on-device storage and offline authentication.

- ▶ Providing mobile application security.

In addition to protecting the on-device data, it is also important to protect the mobile application itself on the device. This prevents hackers from unpackaging a legitimate mobile application and then repackaging it with malicious code:

- Worklight provides an Application Center that can be used to install, configure, and administer a repository of mobile applications for use by individuals and groups within an enterprise or organization (see 6.2, “Worklight Server” on page 52).
- Worklight provides capabilities to encrypt the application code and web resources to prevent tampering with the application.
- By combining multiple authenticity tests (multi-factor authentication), Worklight can enforce more stringent levels of security for the application, device, and user. For example, by requiring application, device, and user authenticity tests, it is possible to only grant access to this legitimate application running on that authorized device for this authenticated user.
- Worklight also extends the concept of SSO to the applications on the mobile device, so that authenticating to one application means that the user does not have to authenticate to other applications on that device.

- ▶ Ensuring security updates.

In today’s mobile world, users can choose whether to download and install the latest release of a mobile application from an application store. It is difficult to ensure that users are downloading and running the correct version of the application in a timely manner. In the event that a fix is needed to correct a security flaw in the application, a timely propagation of security updates is essential to mitigate the possibility of critical problems.

Worklight provides features to help administrators ensure that critical updates are delivered to the applications on the mobile devices.

- ▶ Providing robust authentication and authorization.

Worklight’s authentication integration framework simplifies the task of connecting mobile applications with the enterprise back-end authentication infrastructure. Server-side components interact with the client-side security framework in a challenge-response process to ensure that only authenticated identities are used to access protected resources. Worklight supports a number of commonly used mechanisms for authentication, such as forms-based, cookie-based, or header-based, and so on.

For more information about Worklight security, see the IBM Redbooks publication *Securing Your Mobile Business with IBM Worklight*, SG24-8179.

7.1.5 DataPower security

WebSphere DataPower Appliances simplify, govern, and optimize the delivery of services and applications, and enhance the security of Extensible Markup Language (XML) and information technology (IT) services. In addition to the core business of service-oriented architecture (SOA) connectivity, WebSphere DataPower Appliances now serve areas of business-to-business (B2B) connectivity, web application proxying, and Web 2.0 integration with JSON and Representational State Transfer (REST).

WebSphere DataPower SOA Appliances provide the following key features:

- ▶ Acts as a web, mobile, and XML firewall
- ▶ Enables new workloads for securing mobile, web, and application programming interface (API) management, consolidating and simplifying enterprise infrastructure
- ▶ Provides authentication, authorization, and auditing (AAA) support
- ▶ Provides application-level security as an integral part of the user interaction
- ▶ Helps customers meet compliance requirements, serving as a governance policy enforcement point
- ▶ Can implement an enterprise SSO function using Lightweight Third Party Authentication (LTPA) tokens
- ▶ Simplifies integration to multiple back-end applications (including CICS), supporting a wide array of protocols

DataPower can be used on its own as a mobile gateway to CICS. For example, it can be used to authenticate a mobile user, and map the user's credentials to a security token understood by CICS, a RACF user ID, or an Extended Identity Context Reference (ICRX). An ICRX is a z/OS identity token that contains a distributed identity. When the request is processed by CICS, it resolves the distributed identity to a RACF user ID, and sets the user ID of the CICS task to this value.

The z/OS identity propagation enables a z/OS security administrator to create a set of flexible rules, stored in the RACF database, and ensures that the distributed identity persists after the mapping stage and remains visible for operational support and auditing. For more information about using z/OS identity propagation with CICS see the IBM Redbooks publication *CICS and SOA: Architecture and Integration Choices*, SG24-5466.

DataPower can also be used in conjunction with Worklight. When used with Worklight, it can provide security capabilities beyond those provided by Worklight itself:

- ▶ Enhanced form-based authentication support for easy and quick integration with Worklight applications running on mobile devices
- ▶ Ready-to-use configuration patterns as a reverse proxy and security policy enforcement point in front of the Worklight Server
- ▶ Fine-grained authorization and authentication with a centralized policy enforcement
- ▶ Enhanced data transformation and connectivity capabilities

For more information about the WebSphere DataPower Appliances, see the IBM Redbooks publication, *Strategic Overview of WebSphere Appliances*, REDP-4790.

7.1.6 IBM Endpoint Manager for Mobile Devices

IBM Endpoint Manager for Mobile Devices provides a single platform with complete integration for managing, securing, and reporting on notebooks, desktops, servers, smartphones, tablets, and even point-of-sale terminals. The benefit to the enterprise is visibility and control over all devices, and cost reduction, productivity increases, and compliance improvements.

For more information about IBM Endpoint Manager for Mobile Devices, visit the following website:

<http://www-03.ibm.com/software/products/us/en/ibmendpmanaformobidevi/>

7.2 Configuring security for JSON web services

This section shows how to configure security for CICS JSON web services.

7.2.1 Configuring the URIMAP

A URIMAP resource definition matches the URIs of JSON web service requests. The URIMAP associates a URI for the request with a PIPELINE and WEBSERVICE resource that specifies the processing to be performed.

Importantly, you can use a URIMAP to specify the following attributes:

- ▶ The name of the transaction that CICS uses for running the pipeline alias transaction (the default is CPIH)
- ▶ The user ID under which the pipeline alias transaction runs

Table 7-1 on page 76 shows the attributes of the URIMAP resource definition that affect the security context within which a service provider application runs.

Table 7-1 Security attributes in URIMAP resource for CICS service provider

Attribute	Description
HOST	Specifies the host component of the URI to which the URIMAP definition applies. An example of a host name is www.example.com. This attribute can be used to restrict web service requests to specific host names.
PIPELINE	Specifies the name of the PIPELINE resource definition for the web service. The PIPELINE resource definition provides information about the message handlers, including security message handlers, which act on the service request from the client.
SCHEME	Specifies the scheme component of the URI to which the URIMAP definition applies, which is either HTTP (without SSL) or HTTPS (with SSL). It can be used to restrict web service requests to HTTPS only.
TCIPSERVICE	Specifies the name of a TCIPSERVICE resource definition, that defines an inbound port to which this URIMAP definition relates. It can be used to restrict access to web services through a specific TCIPSERVICE and its associated transport-based security mechanisms.
TRANSACTION	Specifies the name of the pipeline alias transaction that is to be used to start the pipeline. This is an important attribute, because it directly controls the transaction identifiers that are used for web service requests and that, therefore, need to be protected using transaction security.

Attribute	Description
HOST	Specifies the host component of the URI to which the URIMAP definition applies. An example of a host name is www.example.com. This attribute can be used to restrict web service requests to specific host names.
PIPELINE	Specifies the name of the PIPELINE resource definition for the web service. The PIPELINE resource definition provides information about the message handlers, including security message handlers, which act on the service request from the client.
SCHEME	Specifies the scheme component of the URI to which the URIMAP definition applies, which is either HTTP (without SSL) or HTTPS (with SSL). It can be used to restrict web service requests to HTTPS only.
TCPIPSERVICE	Specifies the name of a TCPIPSERVICE resource definition, that defines an inbound port to which this URIMAP definition relates. It can be used to restrict access to web services through a specific TCPIPSERVICE and its associated transport-based security mechanisms.
USAGE	Must specify PIPELINE to indicate that this URIMAP definition applies to inbound web service requests.
USERID	Specifies the user ID under which the pipeline alias transaction is attached. Important: A user ID that you specify in the URIMAP definition is overridden by any user ID that is obtained directly from the client.
WEBSERVICE	Specifies the name of the web service.

Further details about these URIMAP attributes are provided in the CICS Information Center.

When you install a PIPELINE resource, CICS scans the directory specified in the pipeline's WSDIR attribute (the pickup directory) for WSBIND files, and creates URIMAP and WEBSERVICE resources dynamically. If you want to use the URIMAP definition to specify either the name of the transaction or the user ID under which the pipeline will run, you can set these parameters using the DFHLS2JS procedure.

7.2.2 Configuring the TCPIPSERVICE

A TCPIPSERVICE definition is required for a JSON web service that uses HTTP or HTTPS as transport. It contains information about the port on which inbound requests are received, and whether any transport-based security mechanisms will be applied by CICS.

Table 7-2 shows the attributes of the TCPIPService resource definition that affect the security context within which a service provider application runs.

Table 7-2 Security attributes in TCPIPService resource

Attribute	Description
AUTHENTICATE	Determines if an authentication and identification scheme is to be used at the transport layer. Specify BASIC if you want to use HTTP basic authentication, or CERTIFICATE if you want to use SSL client authentication.
CERTIFICATE	Specifies the label of an X.509 certificate that is used as a server certificate during an SSL handshake.
CIPHERS	Specifies the list of ciphers that this CICS region supports for SSL encryption.
PORTNUMBER	Specifies the number of the port on which CICS is to listen for incoming HTTP or HTTPS requests.
SSL	Specifies whether SSL is used for encryption and authentication. If you specify YES, CICS will send a server certificate. If you specify CLIENTAUTH, CICS will request the client to send a certificate.

Further details about these TCPIPService attributes are provided in the CICS Information Center.

Note: HTTP basic authentication scheme can only be considered a secure means of authentication when the connection between the web service client and the CICS region is secure. It is therefore suggested that basic authentication is used in parallel with an SSL connection to protect the user ID and password from being intercepted.

7.3 Worklight security configuration

This section shows how to configure Worklight for secure connectivity to CICS.

7.3.1 Configuring HTTP basic authentication in Worklight

If the TCPIPService definition installed in CICS is configured for HTTP basic authentication, follow these steps to configure the Worklight adapter to send basic authentication credentials:

1. Open the Worklight adapter's XML configuration file and select the design tab. Make sure that each of the procedures has the *connect As* option set to server.
2. Open the source tab to see the XML of the adapter's XML configuration file.

The <connectionPolicy> element is extended to contain an <authentication> child element, as shown in Figure 7-3 on page 79.

```

<wl:adapter name="GENAPPInquire"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wl="http://www.worklight.com/integration"
  xmlns:http="http://www.worklight.com/integration/http">

  <displayName>GENAPPInquire</displayName>
  <description>GENAPPInquire</description>
  <connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>http</protocol>
      <domain>example.com</domain>
      <port>54321</port>

      <authentication>
        <basic/>
        <serverIdentity>
          <username> ${user} </username>
          <password> ${password} </password>
        </serverIdentity>
      </authentication>
    </connectionPolicy>

    <loadConstraints maxConcurrentConnectionsPerNode="2" />
  </connectivity>

```

Figure 7-3 The connectionPolicy element for HTTP basic authentication

The `${user}` can be replaced with a string of your choice or a variable, as per the Worklight Information Center subject in the following link:

http://pic.dhe.ibm.com/infocenter/wrklght/v6r0m0/topic/com.ibm.worklight.help.doc/devref/r_the_connectionpolicy_element.html?resultof=%22%73%73%6c%22%20

7.3.2 Configuring SSL in Worklight

If the TCPIPService definition installed in CICS is configured for SSL, follow these steps to configure the Worklight adapter to use SSL:

1. Open the adapter's XML configuration file and select the design tab (Figure 7-4).

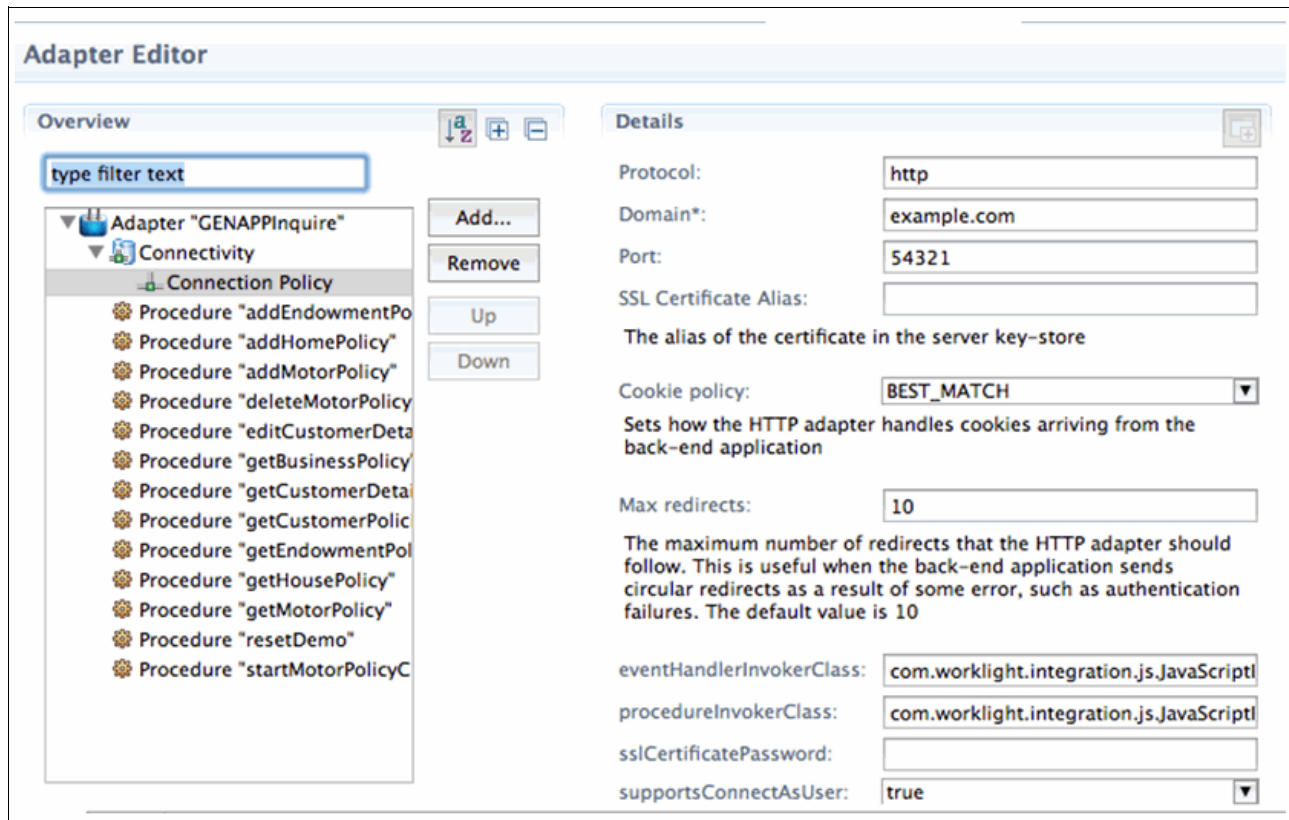


Figure 7-4 Adapter Editor

2. Add the required certificates to the server keystore, as per the procedure detailed in the following link:

http://pic.dhe.ibm.com/infocenter/wrklight/v6r0m0/topic/com.ibm.worklight.help.doc/admin/r_ssl_certificate_keystore_setup.html?resultof=%22%73%73%6c%22%20

3. Add the alias of the SSL Certificate that was added to the server's keystore to the appropriate box. If the certificate is protected with a password, make sure to enter the password in the **sslCertificatePassword** field.
4. To enable SSL for the IBM Worklight application center, follow the procedure shown in the following information center link:

http://pic.dhe.ibm.com/infocenter/wrklight/v6r0m0/topic/com.ibm.worklight.help.doc/appcenter/c_ac_ssl_config.html?resultof=%22%73%73%6c%22%20

7.4 Workload management overview

CICS has been providing web services capabilities since CICS TS V3, when support was introduced for SOAP web services.

The CICS Transaction Server for IBM z/OS Feature Pack for Mobile Extensions V1.0 introduces new capabilities to CICS web services: CICS can now receive and process an HTTP payload in JSON data format.

There are some differences with regard to JSON web service processing when compared with traditional CICS SOAP web services: JSON data is required to be stored within an HTTP payload, because WebSphere MQ transport is not currently supported. Additionally, the CICS pipeline processing is used within the CICS Java virtual machine (JVM) server.

Many of the workload management techniques that apply to SOAP web services also apply to JSON web services. You can take advantage of workload management techniques that might already be in place, such as TCP/IP load balancing and hosting the service provider over multiple listener and application-owning regions.

Note: This chapter summarizes some of the key concepts covered in the chapter on workload management and availability in *CICS Web Services Workload Management and Availability*, SG24-7144, and you should review that information in full for a detailed description of workload management topics.

7.5 Workload balancing

As the service hit rate from mobile applications increases, or due to increased availability demands, it might become necessary to balance a JSON web services workload across multiple CICS regions.

For HTTP, this can be achieved by using port sharing, or the Sysplex Distributor, to route the incoming requests to different CICS regions within a Sysplex. When within CICS, the existing business logic application that is linked from the message adapter can be on an application owning region (AOR) and workload managed, for example, by CICSplex System Manager (CICSplex SM).

Figure 7-5 shows an example of workload balancing across multiple regions.

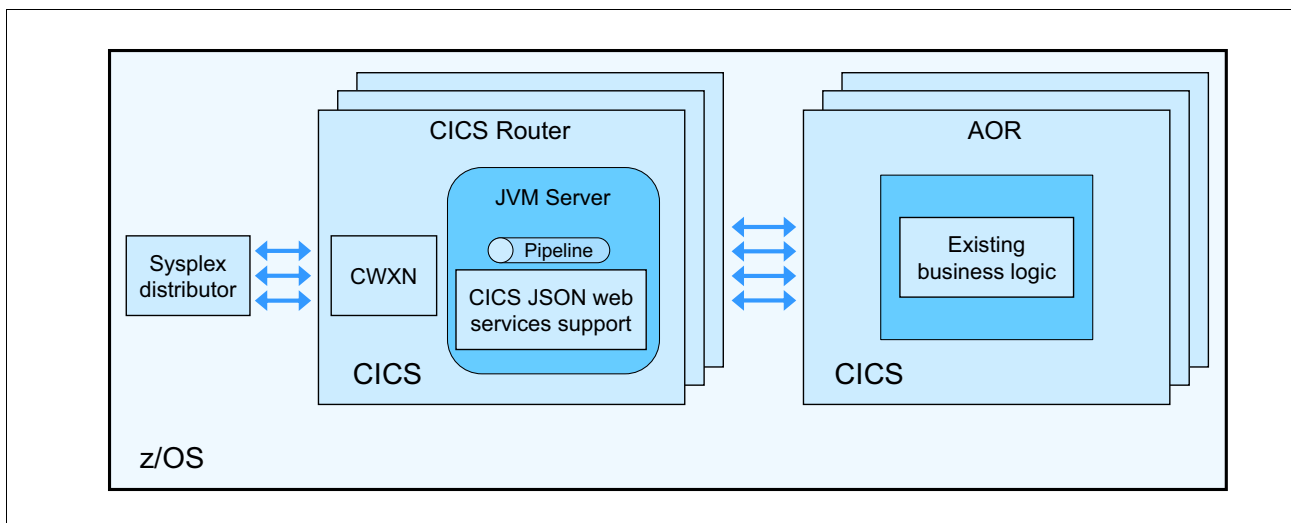


Figure 7-5 Workload balancing across multiple regions

7.6 TCP/IP load balancing techniques

In this section, various TCP/IP load balancing techniques are summarized, describing the attributes of port sharing, virtual IP addressing, and the Sysplex Distributor.

7.6.1 Port sharing

TCP/IP port sharing provides a simple way of spreading HTTP requests over a group of CICS router regions running in the same z/OS image. CICS TCPIPSERVICES in different regions are configured to listen on the same port, and TCP/IP is configured with the SHAREPORT or SHAREPORTWLM options.

The TCP/IP stack then balances connection requests across the CICS router regions. When SHAREPORT is specified on the PORT statement in the TCP/IP profile, TCP/IP evenly balances the number of active connections across the available servers. This balancing is based on the number of active and backlog socket connections.

Using port sharing spreads the JSON request messages across multiple CICS regions, and therefore improves availability. There remains, however, a single point of failure in the event of an IP stack or z/OS image failure.

7.6.2 Virtual IP addressing

A virtual IP address (VIPA) is configured the same way as a normal IP address or a physical adapter, except that it is not associated with any particular interface. TCP/IP hosts can use VIPA (a virtual device and virtual IP address) to select a z/OS IP stack without choosing a specific network interface on that stack. The virtual device defined for the VIPA is always active.

Dynamic VIPA (DVIPA) was introduced to enable the dynamic activation of a VIPA in addition to the automatic movement of a VIPA to another surviving z/OS image after a z/OS stack failure. There are two forms of DVIPA, both of which can be used for takeover functionality:

- ▶ Automatic VIPA takeover enables a VIPA address to move automatically to a stack (called a backup stack) where an existing suitable application instance is already active, and enables the application to serve the client formerly going to the failed stack.
- ▶ DVIPA activation for an application server enables an application to create and activate VIPA so that the VIPA moves when the application moves.

7.6.3 Sysplex Distributor

Sysplex Distributor is designed to address the requirement of one single network-visible IP address for the sysplex cluster. Clients in the network receive the benefits of workload distribution and high availability across the sysplex cluster. With Sysplex Distributor, client connections seem to be connected to a single IP host, even if the connections are established with different servers in the same sysplex cluster.

As with TCP/IP port sharing, Sysplex Distributor also supports server-specific Workload Manager (WLM) recommendations for load balancing. The distribution of new connection requests can now be based on the actual workload of a target server. Sysplex Distributor also takes into account information, such as quality of service (QoS) and policy information, provided by Communications Server for z/OS IP's Service Policy Agent.

By combining this information with the information from WLM, the Sysplex Distributor has the unique ability to ensure that the best destination server instance is chosen for a particular client connection.

Sysplex Distributor has benefits over other load-balancing implementations:

- ▶ Cross-system coupling facility (XRF) links can be used between the distributing stack and target servers, as opposed to LAN connections, such as an Open Systems Adapter (OSA).
- ▶ It provides a total z/OS solution for TCP/IP workload distribution.
- ▶ It provides real-time workload balancing for TCP/IP applications, even if clients cache the IP address of the server, which is a common problem for Domain Name System (DNS)/WLM.
- ▶ It provides for takeover of the VIPA by a backup system if the distributing stack fails.
- ▶ It enables nondisruptive take back of the VIPA original owner to get the workload to where it belongs. The distributing function can be backed up and taken over.

It is possible to combine the use of Sysplex Distributor with TCP/IP port sharing for a high-availability CICS service provider configuration. Then the Sysplex Distributor distributes requests across logical partitions (LPARs), and port sharing distributes requests across different CICS systems within an LPAR.

When CICS is hosting a JSON web service requester application, Sysplex Distributor can also be used to route requests to multiple instances of the service provider, so long as the service provider application runs within the same parallel sysplex as the requester.

7.7 JSON web services and business logic: A multi-region approach

A target business logic application might run in the same CICS region that receives a JSON web service request (the front-end region), or it might run in another region (for example, an AOR). There are several advantages to running the target business logic program in a different region than the one that receives the JSON web service request:

- ▶ You can provide higher availability by having several regions that perform the same business function. If one of the regions fails, other regions of the same group can pick up the workload.
- ▶ You can implement workload balancing and workload separation.
- ▶ You are able to handle increasing workload by adding more CICS AORs.

There are two possible approaches to building a multi-region, front-end/back-end environment.

The simplest approach is to use a distributed program link (DPL) to start the business logic program. All pipeline processing is done in the front-end region, and the link to the business logic program is routed to an AOR. Separation of requests between different AORs might be achieved using workload separation based on different transaction IDs.

It is also possible to route the entire pipeline and business logic processing to an AOR. This might be done by setting the transaction ID in the URIMAP, and then dynamically routing this transaction. Setting the transaction ID in the URIMAP is necessary, because you cannot change the definitions of the default CICS pipeline transaction (CPIH).

Routing using DPL requests is suggested for the following reasons:

- ▶ It establishes a clearer definition, and a separation of the roles of the front-end regions and AORs.
- ▶ It performs better than transaction routing, because it avoids the effect of routing part of the pipeline processing.
- ▶ It is quicker for AOR regions added to the cluster to start processing work, compared to a new listener region that will wait for new HTTP connections. Note that most HTTP clients will continually reuse a pool of connections under steady load. AOR regions can be removed from the cluster more quickly without waiting for clients using HTTP persistent connections to decide to close them.



Problem determination

This chapter outlines common user faults that you might encounter, provides advice for how to avoid these faults, and directions to further information.

This chapter contains the following topics:

- ▶ 8.1, “Introduction” on page 86
- ▶ 8.2, “Deployment problems” on page 86
- ▶ 8.3, “Problems with the JSON assistants” on page 87
- ▶ 8.4, “Problems with requests to JSON web services” on page 88

8.1 Introduction

With the Customer Information Control System (CICS) support for JavaScript Object Notation (JSON) web services being built on the existing web services infrastructure, the approach to take if you encounter problems is largely the same.

Notable however, is the relatively small set of tools available for assisting with the construction of JSON schemas and JSON data, compared to the rich tools available for working with Web Services Description Language (WSDL) and Extensible Markup Language (XML) for SOAP web services.

In addition, the JSON schema standard is still evolving. The CICS JSON assistants are based on draft 4 of the specification. For further information see the following websites:

- ▶ Get information about JSON schema:
<http://json-schema.org/>
- ▶ Get information about JSON schema core definitions and terminology:
<http://tools.ietf.org/html/draft-zyp-json-schema-04>
- ▶ Get information about JSON schema interactive and non-interactive validation:
<http://tools.ietf.org/html/draft-fge-json-schema-validation-00>

As a result of the smaller set of tools, you might find the most common cause of problems to be badly constructed JSON data. Section 8.4, “Problems with requests to JSON web services” on page 88, provides information about common pitfalls and identify tools that can provide assistance.

Tip: You can confirm that the CICS Transaction Server (CICS TS) Feature Pack for Mobile Extensions is working correctly by running the sample program DFH0MOBI. For more information and instructions, see the “Verifying the operations of the CICS TS Feature Pack for Mobile Extensions” topic in the CICS TS Information Center.

If you encounter a problem, before you contact IBM support, see “Collect troubleshooting data for CICS Transaction Server for z/OS Feature Pack for Mobile Extensions V1.0”, found at the following website:

<http://www.ibm.com/support/docview.wss?uid=swg21634271>

This page lists the documentation that you must collect so that the CICS support team can diagnose your problem.

8.2 Deployment problems

Deployment problems are errors that occur when you try to install a PIPELINE resource or a WEBSERVICE resource. As demonstrated in Chapter 9, “Language structure to JSON schema scenario” on page 93, and Chapter 10, “JSON schema to language structure scenarios” on page 105, using the PIPELINE scan operation automatically creates WEBSERVICE and URIMAP resources.

If you are encountering problems after manually defining these resources, the automation of the PIPELINE scan operation might prove easier and less error-prone. For more information about resolving deployment problems, see the section on troubleshooting deployment problems in the CICS TS Information Center.

8.3 Problems with the JSON assistants

This section covers problems that you might encounter when using the JSON assistants. For additional information see the “Troubleshooting the JSON assistant” topic in the CICS TS Information Center.

As presented earlier in this chapter, the JSON assistants are based on draft 4 of the JSON schema specification. This should not be confused with the **MAPPING-LEVEL** parameter in the JSON assistants. This parameter concerns how information is converted between language structures and JSON schema. To benefit from the most sophisticated mappings available, set this to level 3.0.

Tip: With a **MAPPING-LEVEL** of 1.2 or higher, you can use the **CHAR-VARYING** parameter to specify how variable-length character data is mapped between JSON and high-level language structures (and vice versa). For details of usage, see the sub-topic of the “High-level language and JSON schema mapping” topic in the CICS TS Information Center, which applies to the mapping you want to produce.

On multiple occasions, when running DFHJS2LS or DFHLS2JS, we encountered the following DFHPI9523 message:

```
DFHPI9523E An unexpected error occurred whilst processing file
"//USER.JS2LS.COPYLIB(CRREQ01)". The problem is: "//USER.JS2LS.COPYLIB(CRREQ01)".
```

The cause of such messages is usually that the partitioned data set (PDS) cannot be opened because a user has a member of the PDS open for editing, using IBM Rational Developer for IBM System z or Interactive System Productivity Facility (ISPF).

8.3.1 DFHJS2LS

When writing a JSON schema, in preparation for using DFHJS2LS, you might find it helpful to refer to the JSON schema specification and related tutorials. We also suggest that you read the following CICS TS Information Center topics, which describe the subset of the JSON schema specification that is supported:

- ▶ JSON schema to Common Business Oriented Language (COBOL) mapping
- ▶ JSON schema to C and C++ mapping
- ▶ JSON schema to PL/I mapping

We also suggest that you validate your schema. One such tool for validation is the online JSON schema validator:

<http://json-schema-validator.herokuapp.com/>

A JSON schema specifies that a property is optional if it does not appear in the required keyword array that is associated with the enclosing JSON schema object type. As such, DFHJS2LS will add a field, with a suffix `_num`, to generated language structures for optional fields. At run time, this field is set to 1 to indicate that the value was present in the JSON data and set to 0 if it was not.

We suggest that all properties be included in the required keyword array as standard practice. However, CICS does not perform a runtime check for required properties.

Note that CICS cannot transform integer values greater than the maximum value for a signed long ($2^{63} - 1$) unless they are enclosed within quotation marks.

Tip: You can improve the appearance of COBOL language structures generated by DFHJS2LS, by using the **MAPPING-OVERRIDES=UNDERSCORES-AS-HYPHENS** parameter. As a result, DFHJS2LS will use hyphens rather than X characters in variable names in generated high-level language structures.

8.3.2 DFHLS2JS

Before using DFHLS2JS, you might want to read the following CICS TS Information Center topics, which describe the supported mappings and any restrictions that might apply:

- ▶ COBOL to JSON schema mapping
- ▶ C and C++ to JSON schema mapping
- ▶ PL/I to JSON schema mapping

Note: DFHLS2JS does not fully implement the padding algorithms of PL/I. As a result, you might be required to explicitly declare padding bytes. DFHLS2JS will issue a DFHPI9029 or DFHPI9030 message if this is necessary. Further explanation is provided in the PL/I to JSON schema mapping topic, as referenced previously.

8.4 Problems with requests to JSON web services

As introduced at the start of this chapter, the most common problem we have encountered is badly formed JSON data being passed to CICS, which has resulted in the JSON data being rejected, or other unintended results.

It is common to programmatically produce JSON data within your applications. It is suggested that you validate the JSON data produced against the JSON schema as part of the application testing process. The JSON schema validator, referenced in 8.3.1, “DFHJS2LS” on page 87, enables JSON to be validated against a schema.

In addition, ensure that the Multipurpose Internet Mail Extensions (MIME) type of the Hypertext Transfer Protocol (HTTP) requests sent to CICS is `application/json`. This is the official type, as assigned by the Internet Assigned Numbers Authority (IANA).

If you encounter a runtime problem with a JSON web service, you should refer to the following sources of diagnostic information:

- ▶ For web service requests into CICS that fail, first check the HTTP response and status code. The response might contain a CICS message that describes the problem encountered, and perhaps a solution.
- ▶ If the HTTP response and status code does not provide sufficient diagnostic information, further messages might be present in MSGUSR.

In the case of JSON parse errors, which occur when CICS detects that the JSON data received is syntactically invalid, exception messages will be written to the Java virtual machine (JVM) server `stderr`, and to the HTTP response.

Tip: The **WORK_DIR** parameter specified in the JVM profile determines the location of a JVM server’s `stderr` file. If **WORK_DIR** is omitted, `/tmp` is used. For further information, see the “Options for JVMs in a CICS environment” topic in the CICS TS Information Center.

If presented with a CICS message in the range DFHPI1007 - DFHPI1010, a transformation error has occurred. These occur when CICS detects that the JSON data received does not match the expected data format from the WSBIND file.

If you encounter a JSON parse error or transformation error, see any messages issued by CICS, and validate your JSON data against the schema to identify the cause of the problem.

Note that DFHJS2LS produces WSBIND files that describe how to map JSON data to the best fit data types of a particular high-level language. Therefore, CICS does not issue a transformation error message if transformations succeed. This is because the data is within the limits of the high-level language data type, but was outside the range of acceptable values as described in the JSON schema. Such validation should be completed by your mobile application or CICS application.

Because CICS does not perform a runtime check for the existence of required variables, missing numeric fields will be populated with a null value. If uninitialized, such null data might cause a transformation error when the high-level language structure is converted into JSON data for the HTTP response.

For more information, see the topic about troubleshooting problems with JSON requests, found in the CICS TS information center.



Part 3

Application development and scenarios

This part provides information about the application development and tools used in the example scenarios. Highlighted in Part 3, we describe how to call a Customer Information Control System (CICS) JavaScript Object Notation (JSON) service hosted in CICS setup using the Worklight Adapter and the Worklight Client JavaScript application programming interface (API).



Language structure to JSON schema scenario

This chapter describes a scenario that takes an existing Customer Information Control System (CICS) Common Business Oriented Language (COBOL) application, and enables it for use as a JavaScript Object Notation (JSON) web service.

This chapter contains the following topics:

- ▶ 9.1, “General insurance sample application” on page 94
- ▶ 9.2, “Use case for language structure to JSON” on page 94
- ▶ 9.3, “Language Structure to JSON schema solution” on page 94

9.1 General insurance sample application

The scenarios in this IBM Redbooks publication use a general insurance application (GENAPP) available with IBMGENAPP CB12 SupportPac (CB12 SupportPac), which can be found on the following web page:

<http://www.ibm.com/support/docview.wss?uid=swg24031760>

The general insurance application (GENAPP) is a CICS COBOL application that simulates transactions made by an insurance company to create and manage customer and insurance policy data. It provides sample data and an IBM 3270 interface for creating and inquiring about customers and insurance policy information.

The SupportPac documentation describes the application architecture, how to install and set up the application, and how to test the application is working correctly.

For the purposes of the scenarios in Chapter 9, “Language structure to JSON schema scenario” on page 93, and Chapter 10, “JSON schema to language structure scenarios” on page 105, you will require a single CICS region version of the GENAPP. This setup is described in the CB12 SupportPac documentation.

Note: When using a single CICS Region, you do not require the coupling facility named *counter server*, or the *shared temporary storage queue*.

When you have a working application, you can then extend the application to use JSON web services.

9.2 Use case for language structure to JSON

In this scenario, the fictional general insurance company wants to quickly enable a mobile solution for its existing COBOL GENAPP.

The company intends to create an application for mobile devices, implemented in JavaScript and accessing existing CICS and DB2 assets. The initial version of the application will allow customer records to be added to the DB2 database, and for specific records to be queried and updated. It wants to get the solution into the market as soon as possible, and does not currently have the resources to change the existing COBOL application programs.

To resolve the issue, the insurance company is going to enable its existing COBOL applications to use JSON by creating a JSON web service from their existing language structures. In this way the COBOL programs can be left completely unchanged. The following sections of the chapter describe in detail how this will be done using the GENAPP.

9.3 Language Structure to JSON schema solution

Having completed the setup of the GENAPP as described in 9.1, “General insurance sample application” on page 94, you are now in the position to extend the existing GENAPP to make use of JSON web services without changing any of the existing COBOL source or compilations.

This tutorial will configure CICS to enable the following actions:

- ▶ Enable a JSON request to create a customer record.
- ▶ Enable a JSON request to inquire on a customer record.
- ▶ Enable a JSON request to update a customer record.

Perform the following tasks:

1. Identify the general insurance COBOL programs and copybooks to use.
2. Tailor the job control language (JCL) for running DFHLS2JS for the COBOL customer programs.
3. Submit the JCL to create WSBIND files and JSON schemas for each of the listed requests.
4. Set up a PIPELINE to install the WSBIND files and enable a Uniform Resource Identifier (URI) for each request.
5. Test that the JSON request can be successfully performed.

Note: It is assumed that you have set up and installed an appropriate JVMServer and TCPIPSERVICE in your CICS region, as described in Chapter 5, “Configuring CICS for the example scenarios” on page 31.

9.3.1 Identifying the COBOL programs and copybooks

The three requests that this scenario covers (create, inquire, and update a customer record), are handled in the GENAPP by three COBOL programs.

These can be found in the GENAPP source data set:

```
<HLQ>.CB12.SOURCE
```

The following list notes the source code members:

- ▶ LGACUS01 (Customer Create program)
- ▶ LGICUS01 (Customer Inquiry program)
- ▶ LGUCUS01 (Customer Update program)

The GENAPPs are supplied already compiled and installed as programs of the same name in the general insurance load library.

In this scenario, no changes are required to the programs. Instead, you need to create a WEBSERVICE resource that can transform a JSON request to the expected application data.

You need to identify the customer data structure that these programs use to take as input. In three cases the COBOL copybook that they import is LGCMAREA. This copybook is also in the GENAPP source data set.

Looking at the COBOL source code and copybook, you see that the data that you need to be sending to the program is in the CA-CUSTOMER-REQUEST structure, as show in Example 9-1.

Example 9-1 CA-CUSTOMER-REQUEST in LGCMAREA copybook

```
03 CA-REQUEST-ID          PIC X(6).
   03 CA-RETURN-CODE      PIC 9(2).
   03 CA-CUSTOMER-NUM     PIC 9(10).
   03 CA-REQUEST-SPECIFIC PIC X(32482).
*   Fields used in INQ All and ADD customer
   03 CA-CUSTOMER-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
```

05 CA-FIRST-NAME	PIC X(10).
05 CA-LAST-NAME	PIC X(20).
05 CA-DOB	PIC X(10).
05 CA-HOUSE-NAME	PIC X(20).
05 CA-HOUSE-NUM	PIC X(4).
05 CA-POSTCODE	PIC X(8).
05 CA-NUM-POLICIES	PIC 9(3).
05 CA-PHONE-MOBILE	PIC X(20).
05 CA-PHONE-HOME	PIC X(20).
05 CA-EMAIL-ADDRESS	PIC X(100).
05 CA-POLICY-DATA	PIC X(32267).

Note: It is a requirement of the JSON assistants that the data structures are separated from the program code. It also does not support **REDEFINES**.

The GENAPP already provides support for SOAP web services. You can therefore use the supplied data set member S0AIC01 (shown in Example 9-2), which contains the customer request data structure of interest.

Example 9-2 The Customer Request data structure in SOAIC01

01 CA.	
03 CA-REQUEST-ID	PIC X(6).
03 CA-RETURN-CODE	PIC 9(2).
03 CA-CUSTOMER-NUM	PIC 9(10).
* Fields used in INQ All and ADD customer	
03 CA-FIRST-NAME	PIC X(10).
03 CA-LAST-NAME	PIC X(20).
03 CA-DOB	PIC X(10).
03 CA-HOUSE-NAME	PIC X(20).
03 CA-HOUSE-NUM	PIC X(4).
03 CA-POSTCODE	PIC X(8).
03 CA-NUM-POLICIES	PIC 9(3).
03 CA-PHONE-MOBILE	PIC X(20).
03 CA-PHONE-HOME	PIC X(20).
03 CA-EMAIL-ADDRESS	PIC X(100).
03 CA-POLICY-DATA	PIC X(30000).

Having identified the data that the programs require, you look at using this data to generate a JSON schema and a WSBIND file that can be use by a JSON request.

9.3.2 Tailoring DFHLS2JS for the COBOL customer programs

CICS Transaction Server (CICS TS) Feature Pack for Mobile Extensions V1.0 supplies the DFHLS2JS procedure for running the JSON assistants to create a WSBIND file for deployment by your PIPELINE and JSON schemas, which map to the response and requests related to the COBOL data structure.

The DFHLS2JS JCL procedure is found in the Mobile Extensions feature pack installed library, SDFHMOBI. This procedure accepts many parameters, and these are documented in the CICS TS Feature Pack for Mobile Extensions V1.0 Information Center.

For the purpose of this book and this scenario, you will be using a minimum number of parameters to call the DFHLS2JS procedure. In Example 9-3, the JCL needs to be tailored to your environment. Examples of the Inquire Customer and Update Customer assistant JCL are included in Appendix A, “Sample level for a JSON schema” on page 163.

Example 9-3 Sample JCL to run the DFHLS2JS Procedure for Create Customer

```
//LS2JS JOB 'accounting information',name,MSGCLASS=A
//JCLLIB JCLLIB ORDER=CICS51.SDFHMOBI
//LS2JS EXEC DFHLS2JS,
// JAVADIR='java7',
// USSDIR='fp uss dir',
// PATHPREF='',
// TMPDIR='/tmp',
// TMPFILE=''
//INPUT.SYSUT1 DD *
 PDSLIB=GENAPP.CB12.SOURCE
 LANG=COBOL
 MAPPING-LEVEL=3.0
 PGMNAME=LGACUS01
 REQMEM=SOAIC01
 RESPMEM=SOAIC01
 DATETIME=PACKED15
 Log file=/u/cicsuser/genapp/json/logs/LS2JS_LSJSCUSC.LOG
 URI=GENAPP/LSJSCUSC
 PGMINT=COMMAREA
 WSBIND=/u/cicsuser/genapp/json/wsbinding/LSJSCUSC.wsbinding
 JSON-SCHEMA-REQUEST=/u/cicsuser/genapp/json//LGJSCUSCQ.json
 JSON-SCHEMA-RESPONSE=/u/cicsuser/genapp/json/LGJSCUSCR.json
/*
```

Note: The values in **bold** need changing for suitable values when running the DFHLS2JS for the Inquire (LGICUS01) and Update (LGUCUS01) requests. They all use the same SOAIC01 copybook.

The following parameters are supplied:

Log file	The z/OS file system (zFS) file where a log of the DFHLS2JS processing is created.
PDSLIB	The partitioned data set where the language structure source is stored.
PGMNAME	The name of the program that the language structure relates to.
LANG	The high-level language of the language structure source.
MAPPING-LEVEL	The level of mapping applied by the JSON assistant. 3.0 or greater should be used, earlier mapping levels are supported only for compatibility with the SOAP web services assistants.
REQMEM	The copybook in the partitioned data set (PDS) specified by the PDSLIB parameter that the request JSON schema is generated from.
RESPMEM	The copybook in the PDS specified by the PDSLIB parameter that the response JSON schema is generated from.
DATETIME	Specifies if JSON date-time properties in the language structure are mapped as time stamps. PACKED15 indicates that they are mapped as time stamps.

URI	Specifies the relative or absolute URI to be used by the client to access the JSON web service.
PGMINT	Sets how CICS passes the data to the target program.
WSBIND	The zFS file and location of the produced WSBIND file.
JSON-SCHEMA-REQUEST	The zFS location of the JSON schema for the request output.
JSON-SCHEMA-RESPONSE	The zFS location of the JSON schema for the response output.

Full details of all the parameters for DFHLS2JS can be found in the “DFHLS2JS and high-level language to JSON schema conversion for linkable interface” topic in the CICS TS Feature Pack for Mobile Extensions Information Center, which for CICS TS V5.1 is located at the following website:

http://pic.dhe.ibm.com/infocenter/cicsts/v5r1/topic/com.ibm.cics.ts.mobileextensions.doc/reference/dfhws_js21sapi.html

Example JCL for the Customer Inquiry (Example A-1 on page 164) is found in Appendix A, “Sample level for a JSON schema” on page 163.

9.3.3 Submitting the DFHLS2JS JCL

Having tailored the JCL to run the DFHLS2JS, submit the JCL for each of the three programs.

A successful execution of the DFHLS2JS will finish with a return code 0. Verify the job output and log file to resolve any problems, should they occur.

The successful completion will create the following artifacts:

- ▶ A WSBIND file in the location specified to DFHLS2JS.
- ▶ A log file containing diagnostics related to the WSBIND file. You will be asked to supply this file if you need to contact IBM support for assistance.
- ▶ A JSON schema that describes the request to CICS.
- ▶ A JSON schema that describes the response from CICS.

In the case of a language structure to JSON, the produced response and request schemas will often be identical, because the communication area (COMMAREA) will normally be the same. This is the case in this scenario, however the JSON schema would be different, of course, if the copybooks differed.

The key parts of the generated JSON schema for the Customer Create request are shown in Example 9-4. For the full JSON schema that was produced, see “Sample JSON schema generated from COBOL customer create program” on page 164.

Example 9-4 JSON request schema produced from DFHLS2JS for Customer Create

```
{
  "$schema": "http://\json-schema.org/draft-04/\schema#",
  "description": "Request schema for the LGACUS01 JSON interface",
  "type": "object",
  "properties": {
    "LGACUS01operation": {
      "type": "object",
      "properties": {
        "ca": {
          "type": "object",
          "properties": {
```

```

        "ca_request_id":{
            "type":"string",
            "maxLength":6
        },
        "ca_customer_num":{
            "type":"integer",
            "maximum":999999999,
            "minimum":0
        },
        .....
    },
    "required":[
        "ca_request_id",
        "ca_return_code",
        "ca_customer_num",
        .....
    ]
}
},
"required":[
    "ca"
]
}
},
"required":[
    "LGACUS010peration"
]
}
}

```

The JSON produced by the assistant from the COBOL copybook includes all of the data fields that the program requires for input in the JSON schema.

After the schema and description tags, there is the JSON structure itself. The top element in the language structure to JSON generated schema is always a wrapping *operation* field. In the customer create example, this is *LGACUS010peration*.

After the operation field is the JSON representation of the COBOL data structure from the copybook. Because the original copybook has an 01 CA top-level structure, this is mapped to a JSON object, as is the *LGACUS010peration*. In the Customer Create copybook, the data fields that the COBOL program expects are all at level 03. The assistant examines their COBOL data types, and then creates a mapping to a JSON data type.

This results in the COBOL CA-REQUEST-ID field (PIC X(6)) being converted to a JSON string with a maximum length of 6 characters. Conversely, the CA-CUSTOMER-NUM (PIC 9(10)) is mapped to a JSON integer property ranging from 0 - 9999999999.

For more details of language data types and their mappings, see the *CICS TS Feature Pack for Mobile Extensions V1.0* Information center.

In addition, the assistant generates a WSBIND file that is used by CICS to transform the JSON request to the application data.

9.3.4 Enabling the JSON Request URI

To enable CICS to accept JSON requests, for the three customer functions of the GENAPP, the PIPELINE must perform a scan of the WSBIND files.

To perform a scan, you should have already created a PIPELINE resource in a CICS region that has permissions to read the zFS location of the WSBIND files directory specified on the **WSBIND** parameter of the assistants that were run in 9.3.3, “Submitting the DFHLS2JS JCL” on page 98.

Creating a PIPELINE resource is described in Chapter 5, “Configuring CICS for the example scenarios” on page 31.

To perform the PIPELINE scan in CICS Explorer, complete the following steps:

1. Select your PIPELINE resource. Right-click and select **Scan** from the menu as shown in Figure 9-1.

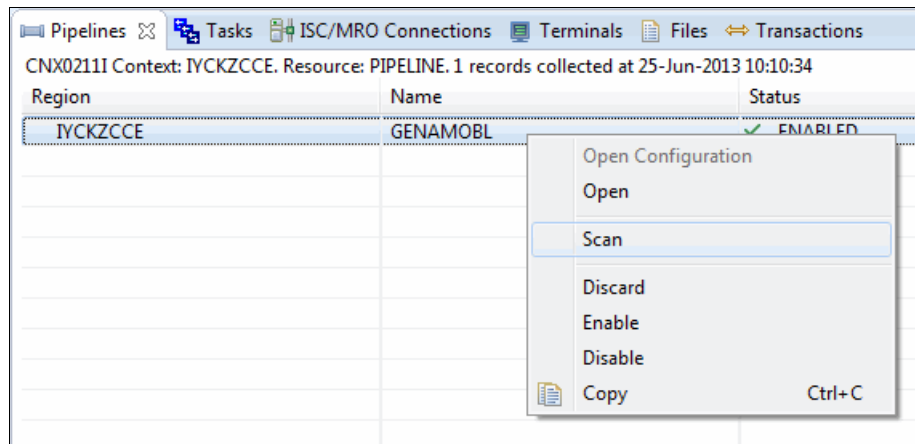


Figure 9-1 IBM CICS Explorer menu for PIPELINE resource

2. When the Perform Scan Operation dialog box appears, click **OK**, as shown in Figure 9-2.

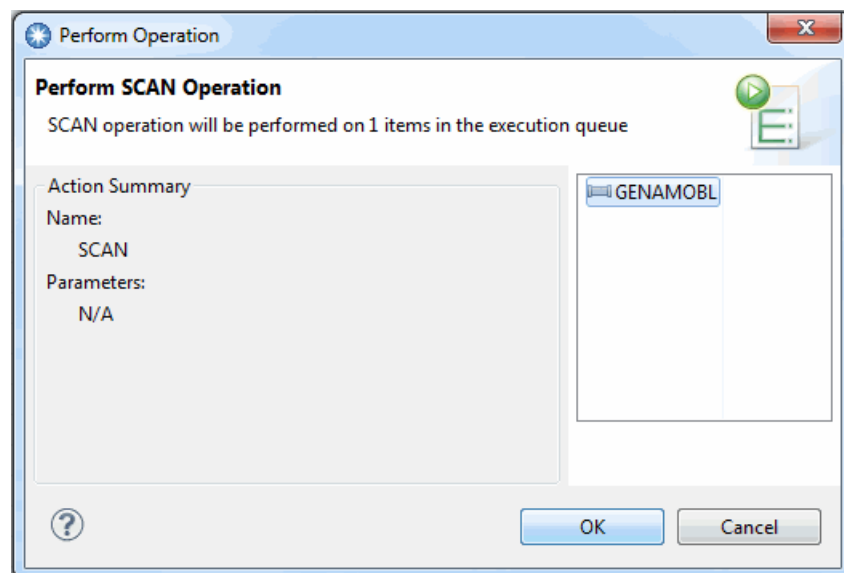


Figure 9-2 IBM CICS Explorer Perform SCAN Operation

The PIPELINE will then scan the pickup directory defined for the resource. This will cause your newly created WSBIND file to be read, and creates the required WEBSERVICE and URIMAP for the JSON web service.

Use CICS Explorer web service operations views to see that these have been created and are in service.

The messages, shown in Example 9-5, are also viewable in the CICS MSGUSR log on a successful PIPELINE scan.

Example 9-5 Example CICS MSGUSR log of PIPELINE Scan success messages

```
DFHPI0703 I 25/06/2013 10:10:34 IYCKZCCE CICSUSER PIPELINE LSTOJSPI is about to
scan the WSDIR directory.
DFHPI0715 I 25/06/2013 10:10:35 IYCKZCCE CICSUSER PIPELINE LSTOJSPI explicit scan
has completed. Number of wsbind files found in the
      WSDIR directory: 000003. Number of WEBSERVICEs created or updated:
000000. Number of WEBSERVICEs not requiring an update:
      000003. Number of failed WEBSERVICE creates or updates: 000000.
```

Having successfully performed the PIPELINE scans, for each of the three WSBIND files, and created the required resources for the JSON requests, CICS can now accept JSON requests for the customer create, inquire, and update functions of the GENAPP.

9.3.5 Test that the JSON request can be successfully performed

To test the JSON request, use the command-line tool **cURL**. It is an open source tool that can be downloaded from the following website:

<http://curl.haxx.se/>

Note: For more information about alternative tools to perform this check, see the following website:

<http://www.json.org/>

To complete this scenario, you make three JSON requests to the GENAPP using **cURL**. Use the following steps to complete this procedure:

1. Send a request to create a customer record. The syntax for **cURL** to send a test JSON request is composed of the following options:
 - The **curl** option, to run the cURL executable file
 - The **-v** option, for verbose (useful for debugging purposes)
 - The **-H** option, to specify the header (use "Content-Type: application/json")
 - The **-d** option, the JSON data to send
 - The URI to send the data to

For the Create Customer request, send some JSON data to the create customer URI.

The data to be sent can be gathered from the JSON schema produced by the assistant. You do not need to send all of the fields in the request, because CICS will populate the fields with blank data where values are not supplied. Then, it is up to the application to handle uninitialized values.

The GENAPP does this and initializes the numeric fields on input. Care must be taken, because failing to handle uninitialized values, and passing this data back, might cause a conversion error on the response. In this scenario, shorten the JSON to just send in the data required by omitting unnecessary string fields.

The data this scenario sent in JSON format is shown in Example 9-6.

Example 9-6 JSON customer create data

```
{"LGACUS010operation":{ "ca" : {  
  "ca_request_id" : "02ACUS",  
  "ca_first_name" : "anew",  
  "ca_last_name" : "customer",  
  "ca_dob" : "1970-01-01",  
  "ca_house_num" : "22",  
  "ca_postcode" : "ZP1 1EX",  
  "ca_email_address" : "example@example.com",  
  }  
}  
}
```

This example will need flattening out on the command line, and the quotations escaped. The command line then looks as per Example 9-7. Change the URI to your CICS JSON customer create URI.

Example 9-7 cURL command line request for create customer request

```
curl -v -H "Content-Type: application/json" -d  
{\"LGACUS010operation\":{\"ca\":{\"ca_request_id\": \"02ACUS\", \"ca_first_name\":  
\"anew\", \"ca_last_name\": \"customer\", \"ca_dob\": \"1970-01-01\", \"ca_house_num  
\": \"22\", \"ca_postcode\": \"ZP11EX\", \"ca_email_address\": \"example@example.  
com\",}}}} http://your.cics.region:30661/GENAPP/LSJSCUSC
```

Running the **cURL** command sends the request, of your JSON request, to the PIPELINE resource. There the data will be transformed to the COBOL program. It processes and creates a new customer record in the DB2 database. Having completed the new customer record task, it will then send a response back to the **cURL** tool with the status of its request and a new customer number. In addition, it includes all of the other customer data described in the JSON responses schema that is produced by the assistant.

Assuming your request was successful, **cURL** should receive an HTTP 200 OK response, application headers, and data (as shown in Example 9-8).

Example 9-8 Example HTTP 200 OK Success Response with headers returned to cURL

```
* About to connect() to your.cics.region:3066 (#0)  
* Trying 256.256.256.256... connected  
> POST /GENAPP/LSJSCUSC HTTP/1.1  
> User-Agent: curl/7.23.1 (x86_64-pc-win32) libcurl/7.23.1 OpenSSL/0.9.8r  
zlib/1  
.2.5  
> Host: winmvs.host.ibm.com:30610  
> Accept: */*  
> Content-Type: application/json  
> Content-Length: 253  
>  
* upload completely sent off: 253 out of 253 bytes  
< HTTP/1.1 200 OK  
< content-type: application/json  
< Date: Tue, 18 Jun 2013 09:43:31 GMT  
< Server: IBM_CICS_Transaction_Server/5.1.0(zOS)  
< Content-Length: 000000000000360  
<
```

```
{"LGACUS010perationResponse":{"ca":{"ca_request_id":"02ACUS","ca_return_code":0,
"ca_customer_num":1000106,"ca_first_name":"anew","ca_last_name":"customer","ca_dob":
"1970-01-01","ca_house_name":"","ca_house_num":22,"ca_postcode":"ZP11EX",
"ca_num_policies":0,"ca_phone_mobile":"","ca_phone_home":"","ca_email_address":
"example@example.com","ca_policy_data":""}}}
```

After the headers, the JSON data is returned. This should include a return code of 0, and the new customer number created by your request.

For the purposes of this scenario, the customer number returned was 1000106. You can then use the customer number in a follow-up request to perform an inquiry on this newly created customer.

2. Send a request to inquire on a customer record.

As before, you can shorten the JSON request, and in this case just supply the two integer fields in the request.

In a more readable format of the JSON, the request this scenario sent is as shown in Example 9-9. Note that the operation field (LGICUS010peration) has changed to the operation field in the generated customer inquiry JSON schema generated by the assistant.

Example 9-9 JSON Customer Inquire data

```
{"LGICUS010peration":{"ca" : {
  "ca_customer_num" : "1000106",
}
}
```

The command line then looks as shown in Example 9-10. Again, change the URI to your CICS JSON customer inquiry URI.

Example 9-10 The cURL command-line request for inquire customer request.

```
curl -v -H "Content-Type: application/json" -d
{"\LGICUS010peration\":{"ca\":{"ca_customer_num\":"1000106\}}}
http://your.cics.region:30661/GENAPP/LSJCSUSI
```

On a successful request, the data to be returned will look as shown in Example 9-11. Again, the request returns all of the required fields in the JSON response schema.

Example 9-11 The JSON data returned by the Inquire Request

```
{"LGICUS010perationResponse":{"ca":{"ca_request_id":"","ca_return_code":0,"ca_
customer_num":1000106,"ca_first_name":"anew","ca_last_name":"customer","ca_dob"
:"1970-01-01","ca_house_name":"","ca_house_num":22,"ca_postcode":"ZP1
1EX","ca_num_
policies":0,"ca_phone_mobile":"","ca_phone_home":"","ca_email_address":"example
@example.com","ca_policy_data":""}}}
```

3. Send a request to update a customer record.

Finally, you send a request to update the customer record. In this example, the customer's house number will be changed from 22 to 42.

You need to send all of the fields back that are populated in the database, because sending blank strings will put a blank string into the customer record on the host database. So the JSON request looks as shown in Example 9-12.

Example 9-12 Customer Update JSON data

```
{"LGUCUS01operation":{ "ca" : {
  "ca_request_id" : "01UCUS",
  "ca_customer_num" : "1000106",
  "ca_first_name" : "anew",
  "ca_last_name" : "customer",
  "ca_dob" : "01-01-1970",
  "ca_house_name" : "",
  "ca_house_num" : "42",
  "ca_postcode" : "ZP11EX",
  "ca_phone_mobile" : "",
  "ca_phone_home" : "",
  "ca_email_address" : "example@example.com",
  "ca_policy_data" : ""
}
}
}
```

Change the URI to match your configuration, using **cURL** command-line format, as shown in Example 9-13.

Example 9-13 The cURL command-line request for update customer request

```
curl -v -H "Content-Type: application/json" -d
{"\LGUCUS01operation\":{"ca\":{"ca_request_id\":"01UCUS\","ca_customer_num\":"0001000106\","ca_first_name\":"anew\","ca_last_name\":"customer\","ca_dob\":"1970-01-01\","ca_house_name\":"\","ca_house_num\":"42\","ca_postcode\":"ZP11EX\","ca_phone_mobile\":"\","ca_phone_home\":"\","ca_email_address\":"example@example.com\","ca_policy_data\":"\"}}}
http://your.cics.region:30661/GENAPP/LSJSCUSU
```

A successful request results in the customer's house number being updated to 42, and the updated record being returned to you.

As can be seen from the JSON that is used in the examples in this chapter, the input structure contains some output-only fields. The reverse might also be true, in that some output data contains input-only data. In addition, the data names are based on the language structure's original names. These names might not be meaningful to the JSON developer.

To make the JSON schema more meaningful to a JSON application developer, it could be modified to suit the service for which it is being used. If the JSON schema is modified, the **WSBIND** file and the **COBOL** structures would need regenerating using **DFHJS2LS**. This would necessitate the creation of a wrapper application to use this new **COBOL** interface with the existing **COBOL** applications. This is described in detail in the scenario in Chapter 10, "JSON schema to language structure scenarios" on page 105.



JSON schema to language structure scenarios

This chapter describes how language structures for Common Business Oriented Language (COBOL), PL/I, C, and C++ can be generated from existing JavaScript Object Notation (JSON) schema definitions. Two approaches for calling existing or new Customer Information Control System (CICS) applications incorporating the generated copybooks are explored in detail.

Procedures for calling CICS applications from the JSON web service Request-Response pattern, and from a Representational State Transfer (REST)-conforming (RESTful) JSON web service interface, are provided.

This chapter describes a process known as the *top-down* scenario, and is of particular relevance where a mobile application is required to interact with a CICS Transaction Server (CICS TS) system from predefined JSON schema definitions.

This chapter contains the following topics:

- ▶ 10.1, “JSON web services: Request-Response and RESTful” on page 106
- ▶ 10.2, “JSON web services: A use case” on page 106
- ▶ 10.3, “Request-Response JSON web service implementation” on page 107
- ▶ 10.4, “RESTful JSON web service implementation” on page 115

10.1 JSON web services: Request-Response and RESTful

The first part of this chapter provides information about implementation aspects of the Request-Response JSON web service pattern. Two JSON schemas, one for request and a second for response, are used as an example case for this model.

This chapter will then build on the RESTful JSON web service information found in earlier chapters to describe an implementation scenario of several RESTful methods.

Note: See the information in 2.4.1, “JSON with feature pack” on page 13, “Request-Response” on page 23, and “RESTful” on page 24, which describe the Request-Response and RESTful JSON web service patterns, and aspects of the CICS implementation of these patterns.

10.2 JSON web services: A use case

In this use case, a company wants to integrate a new mobile application, developed with IBM Worklight, to an existing CICS application, the general insurance application (GENAPP).

The business requirement is for a state of the art mobile application to retrieve customer data from the customer inquiry process contained within GENAPP. In this scenario, the business priority is for a mobile application that is highly intuitive, and that data transfer between the mobile application and CICS is kept to a minimum.

As such, not all of the customer data contained within GENAPP, and stored on a DB2 database, is required by the mobile application. Additionally, given that the GENAPP business logic serves a number of application functions, and that the company has time constraints to deliver the mobile application to the market, no changes are permitted to the base GENAPP application.

A solution to this requirement is for the JSON payload emitted from the mobile application to be processed as a JSON web service by CICS interfacing to GENAPP. This JSON payload could be created from scratch and supplied by the mobile development team, an application design team, or a team of architects. The deliverable is a set of JSON schemas specifying only the data required:

- ▶ One for the request payload
- ▶ One for the response

The JSON schemas are mapped using the DFHJS2LS utility to COBOL language structures. A wrapper program, in this case written in COBOL and deployed to CICS, is used to map the relevant data items in the wrapper program to the format expected by GENAPP. Therefore, no changes are required to GENAPP itself.

The technique described, known as the *top-down* approach, processes existing JSON schema to create traditional language structures. It should be noted that the scenario described interfaces to an *existing* CICS application. However, the top-down approach, given that it creates traditional language structures (such as copybooks and the relevant CICS artifacts) is equally applicable for writing functions in new CICS applications.

The following sections in this chapter describe how this scenario can be implemented by integrating a JSON web service with GENAPP. A step-by-step approach is described, and shows how this business requirement is achieved by both a Request-Response and a RESTful approach.

Figure 10-1 shows how the DFHJS2LS utility processes the JSON schema, and generates both the appropriate language structures and the associated WSBIND file.

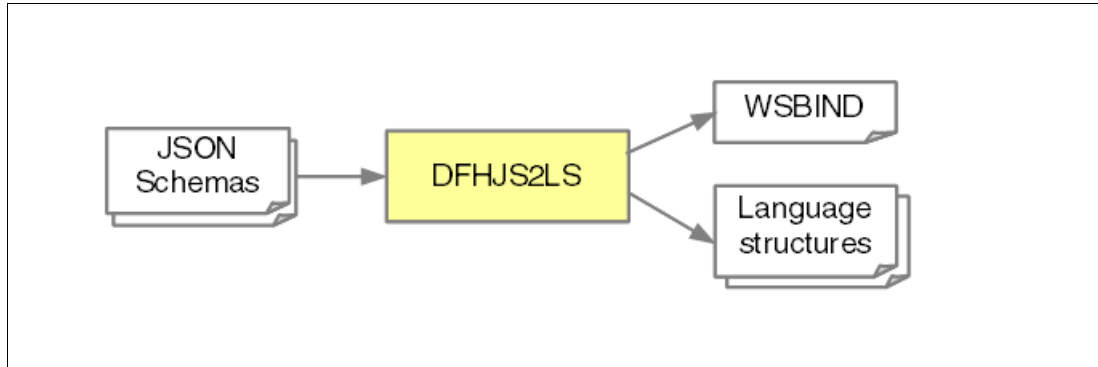


Figure 10-1 DFHJS2LS utility processing

10.3 Request-Response JSON web service implementation

This section details the step-by-step process in implementing the business scenario previously described.

At the end of this section, you will have processed an incoming JSON web service request, potentially arriving from a mobile device, to perform a customer inquiry request to GENAPP. Data returned from the GENAPP customer inquiry request is returned in JSON format.

The following tasks are performed:

1. A review of the incoming and outgoing JSON schema.
2. The definition of the necessary parameters as input to the JSON assistant that maps the JSON schema to the language structure.
3. The development of the CICS wrapper program that performs the transformation between the language structures created by the JSON assistant and the COBOL format required by GENAPP.
4. The definition of the necessary CICS resources, and execution of the PIPELINE scan operation.
5. Testing of the JSON web service to application transformation when starting a GENAPP customer inquiry request.

10.3.1 Reviewing the JSON schema

The mobile development team has created a functional mobile application that is required to interface to the CICS GENAPP application through defined JSON schema interfaces. To facilitate lightweight data transfer, the request schema is required to contain only a couple of elements. This is a key advantage of the JSON schema to language structure mapping process, in that only the JSON elements that are specifically required need to be defined, and can be mapped to a new or existing language structure (for example, a COBOL copybook).

This contrasts with the language structure to JSON schema mapping process, as described in Chapter 9, “Language structure to JSON schema scenario” on page 93, whereby the JSON schema elements are generated for each and every defined data item in the language structure.

Similarly, the response schema contains only the elements required to satisfy the mobile application. Note also that, in terms of naming conventions and data length, there is no direct relationship between the elements defined in the schema and data definition in the existing GENAPP COBOL copybook.

The request JSON schema and an extract of the response JSON schema definitions follow. See Example 10-1 and Example 10-2.

Note: The request and response JSON schema definitions are provided in full in the additional materials that accompany this IBM Redbooks publication.

Example 10-1 Request JSON schema definition

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "description": "JSON request schema for Customer Inquiry",
  "type": "object",
  "properties": {
    "cust_inquiry_request": {
      "type": "object",
      "properties": {
        "function_request_id": {
          "type": "string",
          "minLength": 15,
          "maxLength": 15
        },
        "cust_number": {
          "type": "integer",
          "maximum": 9999999999,
          "minimum": 0
        }
      }
    },
    "required": [
      "function_request_id",
      "cust_number"
    ]
  }
},
"required": [
  "cust_inquiry_request"
]
}
```

Example 10-2 Extract of response JSON schema definition

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "description": "JSON response schema for Customer Inquiry",
  "type": "object",
  "properties": {
```

```

"cust_inquiry_response": {
  "type": "object",
  "properties": {
    "ret_code": {
      "type": "integer",
      "maximum": 999,
      "minimum": 0
    },
    "cust_number": {
      "type": "integer",
      "maximum": 9999999999,
      "minimum": 0
    },
    "first_name": {
      "type": "string",
      "minLength": 20,
      "maxLength": 20
    },
    "last_name": {
      "type": "string",
      "minLength": 20,
      "maxLength": 20
    },
    "date_of_birth": {
      "type": "string",
      "minLength": 10,
      "maxLength": 10
    },
    .....
  }
}

```

When provided by the mobile development team, the request and response JSON schemas are uploaded, using File Transfer Protocol (FTP) or another mechanism, to an appropriate directory in the z/OS UNIX directory structure (z/OS File System, or zFS). For example, this could include the following directories:

```

/u/cicsuser/genapp/json/CustInquiryRequest.json
/u/cicsuser/genapp/json/CustInquiryResponse.json

```

The uploaded JSON schema forms the input to the JSON assistant that performs the mapping between the JSON schema and the language structures.

10.3.2 Mapping the JSON schema to language structures

The CICS TS Feature Pack for Mobile Extensions V1.0 contains JSON assistant utilities that can be used in the creation of service provider applications derived from JSON schema.

Note: See the “Creating a service provider application from a JSON schema” topic in the CICS TS Information Center. It provides a full list of prerequisite configuration information needed before running the JSON assistant utility. This information, for CICS TS 5.1, can be found at the following website:

http://pic.dhe.ibm.com/infocenter/cicsts/v5r1/topic/com.ibm.cics.ts.mobileextensions.doc/reference/dfhws_js2lsapi.html

The DFHJS2LS batch program is supplied in the SDFHMOBI data set supplied with the CICS TS Feature Pack for Mobile Extensions V1.0.

This batch program is used to generate a web service bind file and the appropriate language data structures. DFHJS2LS contains a large set of optional parameters that are fully documented in the CICS TS Information Center.

Example 10-3 shows the batch processing of the JSON schema (previously uploaded), and creates two COBOL copybook files.

Example 10-3 Sample JCL for the DFHJS2LS batch procedure

```
//JS2LSGRR JOB  ,CLASS=A,REGION=900M,
//          MSGCLASS=H,NOTIFY=&SYSUID
//*
//*****
//* JSON to language structure conversion routine
//*****
//*
//JS2LS JCLLIB ORDER=(CICS51.SDFHMOBI)
//  EXEC DFHJS2LS,
//  JAVADIR='java/J7.0_64/J7.0_64',
//  PATHPREF='',
//  USSDIR='uss dir',
//  TMPDIR='/tmp',
//  TMPFILE=''
//INPUT.SYSUT1 DD *
  PDSLIB=USER.JS2LS.COPYLIB
  LANG=COBOL
  MAPPING-LEVEL=3.0
  MAPPING-OVERRIDES=UNDERSCORES-AS-HYPHENS
  PGMINT=COMMAREA
  PGMNAME=GENAJSNW
  REQMEM=JSONRQ
  RESPMEM=JSONRP
  URI=/genapp/CustInquiry
  LOGFILE=/u/cicsuser/genapp/json/logs/CustInquiry.log
  WSBIND=/u/cicsuser/genapp/json/wsbind/CustInquiry.wsbind
  JSON-SCHEMA-REQUEST=/u/cicsuser/genapp/json/CustInquiryRequest.json
  JSON-SCHEMA-RESPONSE=/u/cicsuser/genapp/json/CustInquiryResponse.json
/*
```

Some of the key parameters referenced in the DFHJS2LS utility, shown in Example 10-3, are described in the Table 10-1.

Table 10-1 Key parameters referenced in the DFHJS2LS utility

Parameter	Description
PDSLIB	Specifies the name of the partitioned data set that contains the generated high-level language structure, for example, the generated COBOL copybooks.
LANG	Specifies the programming language of the high-level language structure, for example, COBOL. DFHJS2LS can generate COBOL, C/C++, or PL/I language data structures.
MAPPING-LEVEL	The value of 3.0 should be used as the mapping level to generate JSON schema. MAPPING-LEVEL specifies the level of mapping that DFHJS2LS uses when generating the WSBIND file and language structure.

Parameter	Description
MAPPING-OVERRIDES	Set as UNDERScores-AS-HYPHENS . This parameter value converts any underscores in the Web Services Description Language (WSDL) document to hyphens, rather than the character X, to improve the readability of the generated COBOL language structures.
PGMNAME	Specifies the CICS PROGRAM resource name of the application program, such as the CICS wrapper program that is to be linked to when the service is called.
PGMINT	For a service provider, specifies how CICS passes data to the target application program, either in a CHANNEL or communication area (COMMAREA).
REQMEM	Specifies a 1 - 6 character prefix that DFHJS2LS uses to generate the names of the partitioned data set members that contain the high-level language structures for the web service request, which is the input data to the application program.
RESPMEM	Specifies a 1 to 6 character prefix that DFHJS2LS uses to generate the names of the partitioned data set members that contain the high-level language structures for the web service response, which is the output data from the application program.
URI	This parameter specifies the relative Uniform Resource Identifier (URI) that a client uses to access the web service. CICS uses this when installing the web service as part of a PIPELINE scan operation.
LOGFILE	The fully qualified zFS name of the file into which DFHJS2LS writes its activity log and trace information.
WSBIND	The fully qualified zFS name of the web service bind file to be created.
JSON-SCHEMA-REQUEST	The fully qualified zFS name of the location where the request JSON schema is stored.
JSON-SCHEMA-RESPONSE	The fully qualified zFS name of the location where the response JSON schema is stored.

Note: For more information regarding the JSON to language structure mapping data types, see the “High-level language and JSON schema mapping” topic in the CICS TS Information Center.

10.3.3 Submitting the DFHJS2LS JCL

After configuration of the DFHJS2LS JSON assistant, the job should be submitted to the Job Entry Subsystem (JES) queue, and its return code checked for a successful execution, with a return code value of 0. If a return code value of 0 is not returned, investigate the causes of the failure. In the event of an error, messages are produced in the job log, and they can be a useful source of diagnostic information for further analysis and investigation.

On successful execution, CICS generates the WSBIND file and places it in the location specified by the **WSBIND** parameter.

The COBOL language structures are also created and placed in the partitioned data set specified by the **PDSLIB** parameter, prefixed by the values provided in the **REQMEM** and **RESPMEM** parameters.

An extract of the COBOL copybook, generated as a result of processing the request JSON schema, is shown in Example 10-4.

Example 10-4 Generated DFHJS2LS language structure

06 cust-inquiry-request.	
09 function-request-id	PIC X(15).
09 cust-number	PIC 9(10) DISPLAY.

The data names and the data item-level information contained in the generated copybook can be manually amended to adhere to site standards. However, it is important that no changes to the actual data definitions or order of the data items are made, because this will negate the mapping of the language structures contained within the **WSBIND** file.

10.3.4 Developing the CICS wrapper application program

The COBOL copybooks have been generated and can now be included in a CICS wrapper program. The function of the wrapper program is to map the COBOL data structures defined in the language structure copybooks to a format that is recognizable by an existing CICS application.

At this point, create a new CICS wrapper program, **GENAJSNW**, that will map the generated language structure copybooks into a format that **GENAPP** can process.

An evaluation of how data is passed to the wrapper program is done during the design stage. For example, if the **PGMINT** parameter was set to **COMMAREA** in the **DFHJS2LS** batch procedure, standard **COMMAREA** processing will have to be included in the wrapper program logic. Similarly, if **CHANNEL** was specified as the value for **PGMINT**, channel and container logic should be included in the CICS wrapper program.

Compile the CICS wrapper program using standard compilation procedures, and ensure that the program is in a data set referenced in the **DFHRPL** concatenation or referenced by a **LIBRARY** resource.

Note: The content of the CICS wrapper program, **GENAJSNW** in this example, is provided in the additional materials that accompany this IBM Redbooks publication.

10.3.5 Defining the CICS resources

For the compiled CICS wrapper program, if the CICS autoinstall facility is not used, create a new **PROGRAM** or **LIBRARY** definition for the **GENAJSNW** program using **CICS Explorer**. Define the Program Type as **Assembler**, **C/C++**, **COBOL**, or **PL/I**.

The **DFHJS2LS** JSON assistant generated the **WSBIND** file and placed it in the location specified by the **WSBIND** parameter. The generated web service bind file should be copied to the **pickup** directory of the provider mode **PIPELINE** resource that you want to use for your web service application.

Note: Details about creating a **PIPELINE** configuration can be found in 5.2.2, “How to configure CICS as a service provider” on page 34.

A PIPELINE scan operation should now be performed:

1. Select the appropriate PIPELINE definition in CICS Explorer.
2. Right-click to view options, and select the **SCAN** operation, as shown in Figure 10-2.

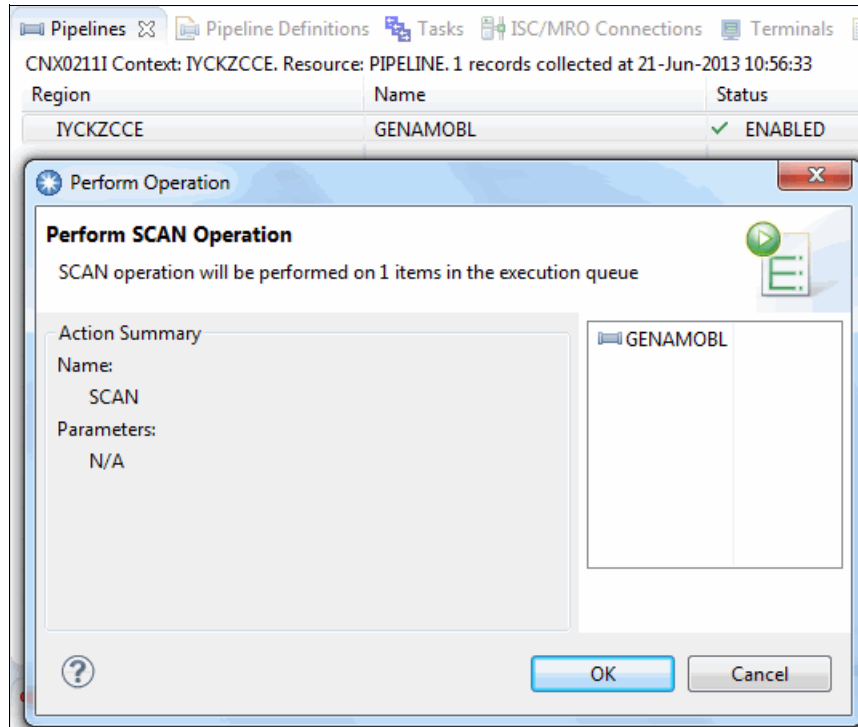


Figure 10-2 Pipeline SCAN operation

The PIPELINE scan operation will dynamically create the WEBSERVICE resource and URIMAP resource. The WEBSERVICE resource encapsulates the web service bind file in CICS, and is used at run time. The URIMAP resource provides CICS with the information to associate the WEBSERVICE resource with a specific URI to accept JSON requests for the GENAPP function.

After the PIPELINE scan operation, validate that the URIMAP and WEBSERVICE resources have been correctly installed into CICS. Using CICS Explorer, use the URI Maps and web service views, as shown in Figure 10-3 and Figure 10-4 on page 114.

Specifically, the CICS WEBSERVICE definition is shown in Figure 10-3.

Region	Name	Pipeline	URI Map	State
IYCKZCCE	CustInquiry	GENAMOBL	\$106020	INSERVICE

Figure 10-3 CICS WEBSERVICE definition

Specifically, the CICS URIMAP definition is shown in Figure 10-4.

Region	Name	Web Service	Path	Pipeline
IYCKZCCE	\$106020	CustInquiry	/genapp/CustInquiry	GENAMOBL

Figure 10-4 CICS URIMAP definition

Note that the name of the WEBSERVICE is derived from the name of the WSBIND file. The path setting, in the URIMAP, is obtained from the **URI** parameter in the DFHJS2LS batch procedure.

Results of the PIPELINE scan operation can also be obtained by viewing the CICS MSGUSR log. Messages will be produced to indicate a successful generation of the WEBSERVICE, or diagnostic information will be produced for further analysis and investigation.

10.3.6 Testing the application

Note: See Chapter 10, section 10.3.5, for details of the **cURL** utility that is used to test the Request-Response JSON web service.

Our application is now ready for testing. To test our scenario, we will send a single function (**getCustomer**) to retrieve customer data from GENAPP. The JSON payload that is sent to CICS for processing is displayed in Example 10-5.

Example 10-5 JSON web service payload for GENAPP customer retrieval

```
{
  "cust_inquiry_request": {
    "function_request_id": "getCustomer",
    "cust_number": 9
  }
}
```

The JSON web service request in Example 10-5 sends the **getCustomer** request to CICS to retrieve customer inquiry data for a specific customer (account number 9).

Because **cURL** is a command-line tool, the command line requires flattening out to the command line and the quotations escaped. The resulting command line is shown in Example 10-6.

Example 10-6 The **cURL** command line for GENAPP customer retrieval

```
curl -v -H "Content-Type: application/json" -X POST -d
{"cust_inquiry_request":{"function_request_id":"getCustomer","cust_number":
:"000000009"}} http://your.cics.region:30661/genapp/CustInquiry
```

Running this command file will send the JSON web service payload to the CICS PIPELINE using the URI specified. The WSBIND file is processed, the JSON web service request is transformed to application data, and the CICS wrapper program is started. The CICS wrapper program maps the COBOL data into a structure that is suitable for processing by GENAPP.

After standard GENAPP processing, in which the appropriate customer information is retrieved, the CICS wrapper program will again convert the GENAPP format data structures into a COBOL format. The COBOL format is transformed to JSON web service data for returning to the **cURL** process.

Successful invocation of the **cURL** command file results in a 200 OK status response, with the customer inquiry data returned from GENAPP, as per Example 10-7.

Example 10-7 Invocation of the CustInquiryRequest command file

```
> POST /genapp/CustInquiry HTTP/1.1
> User-Agent: curl/7.23.1 (x86_64-pc-win32) libcurl/7.23.1 OpenSSL/0.9.8r
zlib/1.2.5
> Host: your.cics.region:30661
> Accept: */*
> Content-Type: application/json
> Content-Length: 89
>
* upload completely sent off: 89 out of 89 bytes
< HTTP/1.1 200 OK
< content-type: application/json
< Date: Mon, 17 Jun 2013 16:04:12 GMT
< Server: IBM_CICS_Transaction_Server/5.1.0(zOS)
< Content-Length: 00000000000183
<
{"cust_inquiry_response":{"ret_code":0,"cust_number":9,"first_name":"Micky","last_
name":"Murphy",
"date_of_birth":"1966-01-03","zipcode":"CA316RN","cell_number":
""},"email_address":""}}* Connection #0 to host your.cics.region left intact
* Closing connection #0
```

Note: The content of the **CustInquiryRequest** command file is provided in the additional materials that accompany this IBM Redbooks publication.

10.4 RESTful JSON web service implementation

This section details the step-by-step process used in implementing the business scenario, described in 10.2, “JSON web services: A use case” on page 106, but this time taking advantage of the RESTful capabilities available with JSON web services.

REST defines a set of architectural principles by which you can design web services that focus on a system's resources, including how resource states are addressed and transferred over Hypertext Transfer Protocol (HTTP).

The example in this book demonstrates some of the key design elements regarding the implementation of a RESTful web service:

- ▶ It uses HTTP methods explicitly.
- ▶ It is stateless.
- ▶ It transfers JSON.

As such, a one-to-one mapping between create, read, and update operations to HTTP methods is used.

This mapping uses the following commands:

POST	To create a resource on the server (a new GENAPP customer).
GET	To retrieve a resource (an existing new GENAPP customer).
PUT	To change the state of a resource or to update it (an existing new GENAPP customer).

At the end of this section, the scenario has processed an incoming JSON web service request, potentially arriving from a mobile device, to perform a number of customer operations. Those operations include customer inquiry request, customer update, and the addition of new customer information. Data returned from these GENAPP customer operations is returned in JSON format.

The following tasks will be performed:

1. Reviewing the JSON schema used for the RESTful operations
2. Defining the necessary parameters as input to the JSON assistant that maps the JSON schema to the language structure for RESTful processing
3. Developing the CICS wrapper program that performs the transformation between the language structures created by the JSON assistant and the COBOL format required by GENAPP
4. Defining the necessary CICS resources, and executing the PIPELINE scan operation
5. Testing the JSON web service to application transformation when starting GENAPP customer operations

10.4.1 Reviewing the JSON schema

The mobile development team has created a functional mobile application that is required to interface to the CICS GENAPP application through a defined JSON schema interface. Unlike the Request-Response scenario, a single JSON schema definition is processed for RESTful processing, and is used for both input and output operations.

An extract of the RESTful JSON schema is included in Example 10-8.

Example 10-8 Extract of the CustService JSON schema for RESTful processing

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "description": "JSON restful schema for Customer Operations",
  "type": "object",
  "properties": {
    "cust_details": {
      "type": "object",
      "properties": {
        "cust_number": {
          "type": "integer",
          "maximum": 999999999,
          "minimum": 0
        },
        "first_name": {
          "type": "string",
          "minLength": 20,
          "maxLength": 20
        },
        "last_name": {
```

```

        "type": "string",
        "minLength": 20,
        "maxLength": 20
    },
    "date_of_birth": {
        "type": "string",
        "minLength": 10,
        "maxLength": 10
    },
    "zipcode": {
        "type": "string",
        "minLength": 8,
        "maxLength": 8
    },
    .....

```

Note: The JSON schema definition for RESTful processing is provided in full in the additional materials that accompany this IBM Redbooks publication

When provided by the mobile development team, the JSON schema for RESTful processing is uploaded, via FTP or another mechanism, to an appropriate directory in the zFS (for example, in the following directory):

```
/u/cicsuser/genapp/json/CustService.json
```

The uploaded JSON schema forms the input to the JSON assistant that performs the mapping between the JSON schema and the language structures.

10.4.2 Mapping the JSON schema to language structures

The CICS TS Feature Pack for Mobile Extensions V1.0 contains JSON assistant utilities that can be used in the creation of service provider applications derived from JSON schema. Artifacts, created by the JSON assistants, apply to both the Request-Response and RESTful JSON web service patterns, and in the development of a JSON web service client application.

Note: See the “Creating a service provider application from a JSON schema” topic in the CICS TS Information Center for a list of prerequisite configuration information before running the JSON assistant.

The DFHJS2LS batch program is provided in the SDFHMOBI data set supplied with the CICS TS Feature Pack for Mobile Extensions V1.0. This batch program is used to generate a web service bind file and the appropriate language data structures. DFHJS2LS contains a large set of optional parameters that are fully documented in the CICS TS Information Center.

The job control language (JCL) shown in Example 10-9 processes the JSON schema for RESTful JSON web service processing that was previously uploaded, and creates a COBOL copybook.

Example 10-9 Sample JCL for the DFHJS2LS batch procedure for RESTful processing

```

//JS2LSGRS JOB , ,CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//*****

```

```

/* JSON schema to wsbind conversion routine
/*****
/*
//JS2LS JCLLIB ORDER=(CICS51.SDFHMOBI)
// EXEC DFHJS2LS,
// JAVADIR='java/J7.0_64/J7.0_64',
// PATHPREF='',
// USSDIR='uss dir',
// TMPDIR='/tmp',
// TMPFILE=''
//INPUT.SYSUT1 DD *
PDSLIB=USER.JS2LS.COPYLIB
PDSMEM=GENRST
MAPPING-LEVEL=3.0
MAPPING-OVERRIDES=UNDERSCORES-AS-HYPHENS
CHAR-VARYING=NO
LANG=COBOL
PGMNAME=GENARSTW
PGMINT=CHANNEL
HTTP-METHODS=GET,POST,PUT
URI=/genapp/CustService/*,
LOGFILE=/u/cicsuser/genapp/json/logs/CustService.log
WSBIND=/u/cicsuser/genapp/json/wsbind/CustService.wsbind
JSON-SCHEMA-RESTFUL=/u/cicsuser/genapp/json/CustService.json
/*

```

Some of the key parameters referenced in the DFHJS2LS batch procedure in Example 10-9 on page 117 are described in Table 10-2.

Table 10-2 Referenced key parameters

Parameter	Description
PDSLIB	Specifies the name of the partitioned data set that contains the generated high-level language structure (for example, the generated COBOL copybooks).
LANG	Specifies the programming language of the high-level language structure (for example, COBOL). DFHJS2LS can generate COBOL, C/C++, or PL/I language data structures.
MAPPING-LEVEL	The value of 3.0 should be used as the mapping level to generate JSON schema. MAPPING-LEVEL specifies the level of mapping that DFHJS2LS uses when generating the WSBIND file and language structure.
MAPPING-OVERRIDES	Set as UNDERSCORES-AS-HYPHENS. This parameter value converts any underscores in the WSDL document to hyphens, rather than the character X, to improve the readability of the generated COBOL language structures.
PGMNAME	Specifies the CICS PROGRAM resource name of the application program, such as the CICS wrapper program that is to be linked to when the service is called.
PGMINT	For a service provider, specifies how CICS passes data to the target application program, either in a CHANNEL or COMMAREA.

Parameter	Description
PDSMEM	Specifies a 1 - 6 character prefix that DFHJS2LS uses to generate the name of the partitioned data set member that contains the high-level language structures for the web service request, which is the input data to the application program.
HTTP-METHODS	If a value is provided, DFHJS2LS builds a WSBIND file in which only the explicitly specified HTTP methods are accepted. The default value is for GET , POST , PUT , and DELETE to be set, which tells DFHJS2LS that the application supports the four main RESTful operations.
URI	This parameter specifies the relative URI that a client uses to access the web service. CICS uses this when installing the web service as part of a PIPELINE scan operation.
LOGFILE	The fully qualified zFS name of the file into which DFHJS2LS writes its activity log and trace information.
WSBIND	The fully qualified zFS name of the web service bind file to be created.
JSON-SCHEMA-RESTFUL	The fully qualified zFS name of the location where the RESTful JSON schema is stored.

Note: For more information regarding the JSON to language structure mapping data types, see the “High-level language and JSON schema mapping” topic in the CICS TS Information Center.

10.4.3 Submitting the DFHJS2LS JCL

After configuration of the DFHJS2LS JSON assistant utility, the job should be submitted to the JES queue, and its return code checked for a successful execution with a return code value of 0. If a return code value of 0 is not returned, investigate the causes of the failure. In the event of an error, messages are produced in the job log, and they can be a useful source of diagnostic information for further analysis and investigation.

On successful execution, CICS generates the WSBIND file and places it in the location specified by the **WSBIND** parameter. The COBOL language structures are also created, and are placed in the partitioned data set specified by the **PDSLIB** parameter, prefixed by the value provided in the **PDSMEM** parameter.

An extract of the COBOL copybook generated as a result of processing the RESTful JSON schema is included in Example 10-10.

Example 10-10 Extract of DFHJS2LS generated COBOL copybook for RESTful processing

```

06 cust-details.
   09 cust-number          PIC 9(10) DISPLAY.
   09 first-name           PIC X(20).
   09 last-name            PIC X(20).
   09 date-of-birth        PIC X(10).
   09 zipcode              PIC X(8).
   09 cell-number          PIC X(20).
   09 email-address        PIC X(40).

```

The data names and the data item-level information contained in the generated copybook can be manually amended to adhere to site standards. However, it is important that no changes to the actual data definitions or order of the data items are made, because this negates the mapping of the language structures contained within the WSBIND file.

10.4.4 Developing the CICS wrapper application program

The COBOL language structure copybook for RESTful processing was created, and it can now be included in a CICS wrapper program. The function of the wrapper program is to map the COBOL data structures defined in the language structure copybooks to a format that is recognizable by a new or existing CICS application.

A new CICS wrapper program, GENARSTW, is created, and it maps the generated language structure copybook into a format that GENAPP can process. GENARSTW then processes inbound JSON web service requests that are started using RESTful method formats. Note that in the sample GENARSTW CICS wrapper program (Example 10-11), the **HEAD** and **DELETE** RESTful methods are not supported.

Example 10-11 Code sample from the GENARSTW COBOL wrapper program

```
*****
* Perform the method
*****
PROCESS-METHOD.
  EVALUATE WS-HTTP-METHOD
    WHEN METHOD-GET
      PERFORM GET-DATA
    WHEN METHOD-PUT
      PERFORM PUT-DATA
    WHEN METHOD-POST
      PERFORM POST-DATA
    WHEN OTHER
      EXEC CICS ABEND
        ABCODE(UNSUPPORTED-METHOD-ABCODE)
      END-EXEC
  END-EVALUATE.
```

Compile the CICS wrapper program using standard compilation procedures, and ensure that the program is in a data set referenced in the DFHRPL concatenation, or referenced by a LIBRARY resource.

Note: The content of the CICS wrapper program, GENARSTW in this example, is provided in the additional materials that accompany this IBM Redbooks publication.

10.4.5 Defining the CICS resources

If the CICS autoinstall facility is not used for the compiled CICS wrapper program, create a new PROGRAM or LIBRARY definition for the GENAJSNW program using CICS Explorer. Define the Program Type as Assembler, C/C++, COBOL, or PL/I.

The DFHJS2LS JSON assistant generates the WSBIND file and places it in the location specified by the **WSBIND** parameter. The generated web service bind file should be copied to the pickup directory of the provider mode PIPELINE resource that you want to use for your web service application.

Note: Details about creating a PIPELINE configuration can be found in the section *How to configure CICS as a service provider* in Chapter 5, “Configuring CICS for the example scenarios” on page 31.

A PIPELINE scan operation should now be performed:

1. Select the appropriate PIPELINE definition in CICS Explorer.
2. Right-click to view options.
3. Select the **SCAN** operation, as shown in Figure 10-5.

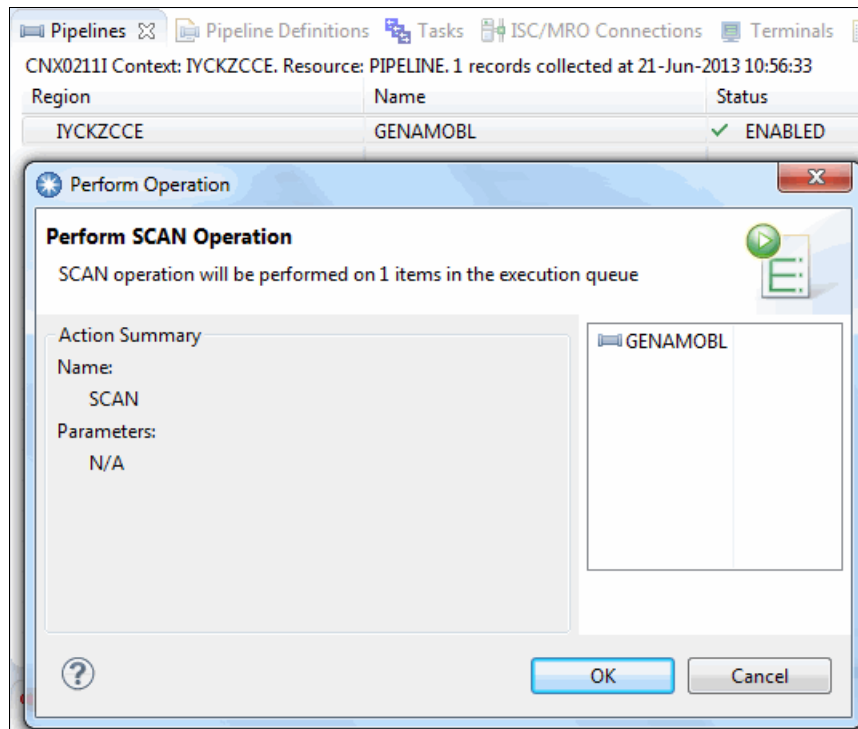


Figure 10-5 Pipeline scan operation

The PIPELINE scan operation will dynamically create the WEBSERVICE resource and URIMAP resource. The WEBSERVICE resource encapsulates the web service bind file in CICS, and is used at run time. The URIMAP resource provides CICS with the information to associate the WEBSERVICE resource with a specific URI to accept JSON requests for the GENAPP function.

After the PIPELINE scan operation, validate that the URIMAP and WEBSERVICE resources have been correctly installed to CICS. Using CICS Explorer, use the URI Maps and web service views as per Figure 10-6 and Figure 10-7 on page 122.

Specifically, CICS WEBSERVICE is shown in Figure 10-6.

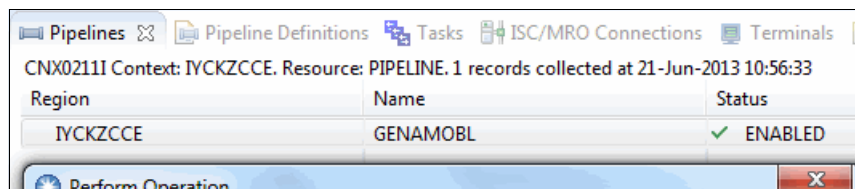


Figure 10-6 CICS WEBSERVICE definition

Specifically, CICS URIMAP is shown in Figure 10-7.

Region	Name	Pipeline	URI Map	State
IYCKZCCE	CustService	GENAMOBL	\$106021	INSERVICE

Figure 10-7 CICS URIMAP definition

Note that the name of the WEBSERVICE is derived from the name of the WSBIND file. The path setting in the URIMAP is obtained from the **URI** parameter in the DFHJS2LS batch procedure.

Results of the PIPELINE scan operation can also be obtained by viewing the CICS MSGUSR log. Messages are produced to indicate a successful generation of the WEBSERVICE, or diagnostic information is produced for further analysis and investigation.

10.4.6 Testing the application

Note: See 9.3.5, “Test that the JSON request can be successfully performed” on page 101, for details of the **cURL** utility that is used to test the RESTful JSON web service.

The application is now ready for testing. To test the scenario, you send various RESTful functions to process GENAPP from various **cURL** command files.

RESTful customer inquiry function

The JSON web service payload is sent to CICS for processing to start the customer inquiry function as a RESTful service.

Because **cURL** is a command-line tool, the command line requires flattening out to the command line and the quotations escaped. The resulting command line is displayed in Example 10-12.

Example 10-12 The cURL command line for the CustServiceREST_GET command file

```
curl -v -H "Content-Type: application/json" -X GET
http://your.cics.region:30661/genapp/CustService/000000009
```

The JSON web service request (Example 10-12 on page 122) sends a single function to CICS to retrieve customer inquiry data for a specific customer (account number 9).

The CICS wrapper program obtains the customer number by accessing CICS containers available for processing web services, such as DFHWS-URIMAPPATH. The code sample in Example 10-13 demonstrates an access of one such container.

Example 10-13 Sample DFHWS-URIMAPPATH container access

```
*****
* Get containers
*****
GET-RESID.
  MOVE ' ' TO WS-RESID
  EXEC CICS GET CONTAINER('DFHWS-URIMAPPATH')
        INTO(WS-RESID)
```

```
RESP(RESP)
RESP2(RESP2)
END-EXEC
```

Note, in keeping with RESTful processing convention, the service name, `CustService`, is generic, and the operation to be performed, `GET`, is specified as an HTTP method. This supports reuse of the JSON web service.

Running this command file sends the JSON web service payload to the CICS PIPELINE using the URI specified. The `WSBIND` file is processed, the JSON web service request is transformed to application data, and the CICS wrapper program is started. The CICS wrapper program maps the COBOL data into a structure that is suitable for processing by GENAPP.

After standard GENAPP processing, in which the customer account information is retrieved, the CICS wrapper program converts the GENAPP format data structures into a COBOL format. That format can be mapped to JSON web service data for returning to the `cURL` process.

Successful invocation of the `cURL` command file results in a 200 OK status response, with the customer inquiry data returned from GENAPP, as shown in Example 10-14.

Example 10-14 Invocation of the `CustServiceREST_GET` command file

```
> GET /genapp/CustService/0000000009 HTTP/1.1
> User-Agent: curl/7.23.1 (x86_64-pc-win32) libcurl/7.23.1 OpenSSL/0.9.8r zlib/1
.2.5
> Host: your.cics.region:30661
> Accept: */*
> Content-Type: application/json
>
< HTTP/1.1 200 OK
< content-type: application/json
< Date: Wed, 19 Jun 2013 12:38:35 GMT
< Server: IBM_CICS_Transaction_Server/5.1.0(zOS)
< Content-Length: 00000000000161
<
{"cust_details":{"cust_number":9,"first_name":"Micky","last_name":"Murphy","date
_of_birth":"1966-01-03","zipcode":"CA316RN","cell_number":"","email_address":""}
}* Connection #0 to host your.cics.region left intact
* Closing connection #0
```

Note: The content of the `CustServiceREST_GET` command file, for running in a Microsoft Windows environment, is provided in the additional materials that accompany this IBM Redbooks publication.

RESTful customer update function

The JSON web service payload, which is sent to CICS for processing to start the customer update function as a RESTful service, is displayed in Example 10-15.

Example 10-15 JSON web service payload

```
{
  "cust_details": {
    "first_name": "James",
    "last_name": "Smith",
```

```

    "date_of_birth": "2001-01-01",
    "zipcode": "SO212JN",
    "cell_number": "07756576667",
    "email_address": "james.smith@anycompany.com"
  }
}

```

Because **cURL** runs on a command-line basis, the JSON web service payload, in Example 10-15, requires flattening out to the command line and the quotations escaped. The resultant command line is displayed in Example 10-16.

Example 10-16 The cURL command line for the CustServiceREST_PUT command file

```

curl -v -H "Content-Type: application/json" -X PUT -d
{"cust_details\":{\"first_name\":\"James\",\"last_name\":\"Smith\",\"date_of_birt
h\":\"2001-01-01\",\"zipcode\":\"SO212JN\",\"cell_number\":\"07756576667\",\"email
_address\":\"james.smith@anycompany.com\"}}
http://your.cics.region:30661/genapp/CustService/0000000009

```

The JSON web service request, shown in Example 10-16, sends a request to CICS to update customer inquiry data for a specific customer (account number 9).

Note, in keeping with RESTful processing convention, that the service name, **CustService**, is generic, and the operation to be performed, **PUT**, is specified as an HTTP method. This enables reuse of the web service.

Running this command file sends the JSON web service payload to the CICS PIPELINE using the URI specified. The **WSBIND** file is processed, the JSON web service request is transformed to application data, and the CICS wrapper program is started. The CICS wrapper program maps the COBOL data into a structure that is suitable for processing by **GENAPP**.

After standard **GENAPP** processing, in which the customer account information is updated, the CICS wrapper program converts the **GENAPP** format data structures into a COBOL format. This format can be mapped to JSON web service data for returning to the **cURL** process.

Successful invocation of the **cURL** command file results in a 200 OK status response, as shown in Example 10-17.

Example 10-17 Invocation of the CustServiceREST_PUT command file

```

> PUT /genapp/CustService/0000000009 HTTP/1.1
> User-Agent: curl/7.23.1 (x86_64-pc-win32) libcurl/7.23.1 OpenSSL/0.9.8r zlib/1
.2.5
> Host: your.cics.region:30661
> Accept: */*
> Content-Type: application/json
> Content-Length: 182
>
* upload completely sent off: 182 out of 182 bytes
< HTTP/1.1 200 OK
< content-type: application/json
< Date: Wed, 19 Jun 2013 14:45:08 GMT
< Server: IBM_CICS_Transaction_Server/5.1.0(zOS)
< Content-Length: 00000000000100
<
/genapp/CustService/0000000009

```

* Connection #0 to host your.cics.region left intact

* Closing connection #0

Note: The content of the `CustServiceREST_PUT` command file, for running in a Microsoft Windows environment, is provided in the additional materials that accompany this IBM Redbooks publication.

The application has returned a URI format structure in the following format:

```
/genapp/CustService/000000009
```

This was returned to the application after being placed in the `DFHRESPONSE` container. This URI can now be processed by the application for additional function.

RESTful customer addition function

The JSON web service payload, which is sent to CICS for processing to start the customer addition function as a RESTful service, is displayed in Example 10-18.

Example 10-18 JSON web service payload

```
{
  "cust_details": {
    "first_name": "James",
    "last_name": "Smith",
    "date_of_birth": "2001-01-01",
    "zipcode": "SO212JN",
    "cell_number": "07756576667",
    "email_address": "james.smith@anycompany.com"
  }
}
```

Because `cURL` runs on a command-line basis, the JSON web service payload, shown in Example 10-18 on page 125, requires flattening out to the command line and the quotations escaped. The resulting command line for processing `cURL` is displayed in Example 10-19.

Example 10-19 The cURL command line for the CustServiceREST_POST command file

```
curl -v -H "Content-Type: application/json" -X POST -d
{"cust_details":{"first_name":"James","last_name":"Smith","date_of_birt
h":"2001-01-01","zipcode":"SO212JN","cell_number":"07756576667","email
_address":"james.smith@anycompany.com"}}
http://your.cics.region:30661/genapp/CustService/
```

The JSON web service request, shown in Example 10-19, sends the request to CICS to add new customer data for a new specific customer. GENAPP returns a new account number.

In keeping with RESTful processing convention, the service name, `CustService`, is generic, and the operation to be performed, `POST`, is specified as an HTTP method.

Running this command file sends the JSON web service payload to the CICS PIPELINE using the URI specified. The `WSBIND` file is processed, the JSON web service request is transformed to application data, and the CICS wrapper program is started. The CICS wrapper program maps the COBOL data into a structure that is suitable for processing by GENAPP.

After standard GENAPP processing, in which the customer account information is added, the CICS wrapper program converts the GENAPP format data structures into a COBOL format. That format can be mapped to JSON web service data for returning to the **cURL** process.

Successful invocation of the **cURL** command file results in a 200 OK status response, with the new customer data returned from GENAPP, as shown in Example 10-20.

Example 10-20 Invocation of the CustServiceREST_POST command file

```
> POST /genapp/CustService/ HTTP/1.1
> User-Agent: curl/7.23.1 (x86_64-pc-win32) libcurl/7.23.1 OpenSSL/0.9.8r zlib/1
.2.5
> Host: your.cics.region:30661
> Accept: */*
> Content-Type: application/json
> Content-Length: 183
>
* upload completely sent off: 183 out of 183 bytes
< HTTP/1.1 200 OK
< content-type: application/json
< Date: Wed, 19 Jun 2013 12:56:02 GMT
< Server: IBM_CICS_Transaction_Server/5.1.0(zOS)
< Content-Length: 000000000000100
<
/genapp/CustService/0001000165
      * Connection #0 to host your.cics.region left intact

* Closing connection #0
```

The new customer account number, assigned as part of GENAPP customer addition logic, is returned to the JSON web service client program in the DFHRESPONSE container, potentially for further processing.

Note: The content of the **CustServiceREST_POST** command file, for running in a Microsoft Windows environment, is provided in the additional materials that accompany this IBM Redbooks publication.



Developing a simple JSON web service client application

This chapter describes how to develop a Customer Information Control System (CICS) application that acts as a client for a JavaScript Object Notation (JSON) web service. To demonstrate this, the chapter walks you through an example application that calls another company's service to retrieve a credit score for a customer.

The solution is described in the following sections:

- ▶ 11.1, "Overview of the solution" on page 128
- ▶ 11.2, "Writing the JSON schema" on page 130
- ▶ 11.3, "Generating the language structures" on page 135
- ▶ 11.4, "Defining the CICS resources" on page 137
- ▶ 11.5, "Developing the application program" on page 142
- ▶ 11.6, "Testing the sample application" on page 148

11.1 Overview of the solution

This section gives an overview of how the solution is implemented, and presents some background information about the linkable interface used to transform JSON.

11.1.1 The scenario

Fictional Insurance Company (company example) wants to better understand the level of risk associated with new motor insurance policies. They have partnered with *Nonexistent Credit Agency (company example)* to obtain insurance-related credit scores for their prospective customers to do this.

Obtaining a credit score for their prospective customers will enable them to gauge the potential level of risk and adjust the quoted premium accordingly. Rather than requiring their staff to call Nonexistent Credit Agency when processing a new policy, they want to take advantage of the JSON web service provided by Nonexistent Credit Agency. This service enables partners to send a JSON request to obtain a credit score.

11.1.2 The solution

As described briefly in 4.2.2, “How CICS supports acting as a client for JSON web services” on page 26, you can develop CICS applications that act as a client for JSON web services using **WEB** application programming interface (API) commands and the linkable interface to transform JSON.

You must create JSONTRANSFRM bundle parts using the JSON assistants to describe the mappings between JSON and application data. The application sends requests to the service using **WEB** API commands, and uses the linkable interface to transform request and response data. Figure 11-1 gives a conceptual view of how data flows between the application and the target service.

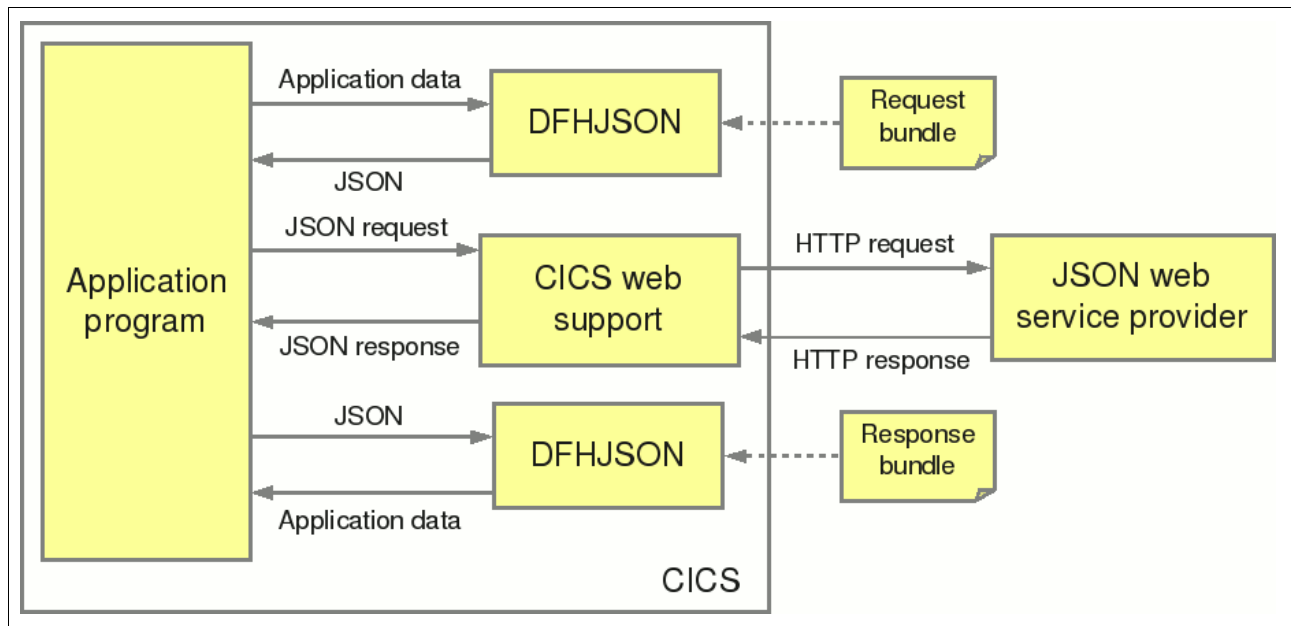


Figure 11-1 Conceptual view of a JSON web service client application

This chapter walks you through creating your JSONTRANSFRM, starting from the interface to Nonexistent Credit Agency's service credit scoring service. Unfortunately, Nonexistent Credit Agency does not provide a JSON schema describing the interface, so your first task is to create one. You can then generate the JSONTRANSFRM bundle, develop the CICS application, and deploy the artifacts.

To demonstrate how the client application works, there is a complete sample program that calls the credit score service and writes information from the response to the terminal. It contains static input data for a fictional customer. In a complete application, such a program can instead be called from the business logic that creates an insurance quote. The Common Business Oriented Language (COBOL) source for the client program is provided in Appendix B, "Sample COBOL programs" on page 167.

To test the client application, there is an implementation of the credit scoring service that runs in CICS. This is a Request-response style JSON web service that was developed from the JSON schema for the service. It simply returns a random credit score that will vary depending on the customer's house number and policy type.

The complete source for the provider program is given in Appendix B, "Sample COBOL programs" on page 167. No further details about the implementation of the service are provided in this IBM Redbooks publication. For details about how to set up client and provider applications, see 11.6, "Testing the sample application" on page 148.

11.1.3 The linkable interface for transforming JSON

The CICS Transaction Server (CICS TS) Feature Pack for Mobile Extensions V1.0 provides a transformer program, DFHJSON, which can be called from an application using a **LINK PROGRAM** command. You can use it to transform application data to JSON, and JSON to application data. Parameters are passed to the transformer using a set of containers that the application must create before calling the transformer, and data is returned to the application in containers.

The transformations between application data and JSON data are described by a JSONTRANSFRM bundle part. The transformation is performed in a Java virtual machine (JVM) server, which must have the `JAVA_PIPELINE=YES` option in the JVM profile. For more information about configuring the JVM server, see Chapter 5, "Configuring CICS for the example scenarios" on page 31.

The JSONTRANSFRM bundle part and the JSBIND file

A JSONTRANSFRM bundle part is generated by the CICS JSON assistants. It describes a single mapping between a language structure and a JSON schema, which can be used at run time to transform application data to JSON and JSON to application data. The JSON assistants generate a CICS bundle on z/OS file system (zFS) containing a JSONTRANSFRM bundle part and a JSBIND file that describes the mapping.

You create a BUNDLE resource pointing to the zFS location of the bundle, and then install it into CICS. You can generate a JSBIND file starting either from a language structure (using DFHLS2JS), or from a JSON schema (using DFHJS2LS).

Containers used with the linkable interface

Before calling the transformer program, the application must create a set of containers that hold the input data, the name of JSONTRANSFRM, and optionally the name of the JVM server where the transformation will be performed. Before the transformer returns control to the application, the transformed data is placed in a container.

If an error occurs during transformation, the transformer creates containers giving details of the error. Table 11-1 gives details of the containers.

Note: The transformer uses the presence of the DFHJSON-DATA or DFHJSON-JSON containers to determine which type of transformation to perform. Therefore, only one of these containers can be present when the transformer is called.

Table 11-1 Containers used with the linkable interface

Container name	Type	Created	Contents
DFHJSON-TRANSFRM	CHAR	By the application	The name of the JSONTRANSFRM bundle part.
DFHJSON-JVMSEVR	CHAR	Optionally by the application	The name of the JVM server that performs the transformation. If this container is not present, the CICS-supplied JVMSEVER DFH\$AXIS is used.
DFHJSON-ERROR	BIT	By CICS if an error occurs	A fullword binary value indicating the type of error that occurred.
DFHJSON-ERRORMSG	CHAR	By CICS for some errors	Further details of the error.
When transforming application data to JSON			
DFHJSON-DATA	BIT	By the application	Application data to be transformed.
DFHJSON-JSON	CHAR	By CICS	JSON corresponding to the application data provided.
When transforming JSON to application data			
DFHJSON-JSON	CHAR	By the application	JSON to be transformed.
DFHJSON-DATA	BIT	By CICS	Application data corresponding to the JSON provided.

11.2 Writing the JSON schema

The first step to create a client application for a JSON web service is to describe the interface made available by that service. This can be done starting with either a JSON schema or a language structure. As the most common scenario is that the service you want to call already exists, this chapter demonstrates starting from a JSON schema. The CICS JSON assistants use information in the JSON schema to map JSON properties to high-level language data types. These mappings are also used at run time to transform between JSON and application data.

Nonexistent Credit Agency does not provide a JSON schema describing the interface to their credit scoring service, but instead provides the following documentation, shown in Example 11-1 on page 131. If a JSON schema is already available that describes the service you want to call, you can skip to 11.3, “Generating the language structures” on page 135.

Example 11-1 Documentation for Nonexistent Credit Agency's insurance scoring service

To request an insurance credit score for an individual, send an HTTP POST request to the following URI:

`http://services.nonexistentcreditagency.com/insuranceScore`

with a JSON body like this:

```
{
  "insuranceScoreRequest": {
    "firstName": "Joe",
    "lastName": "Bloggs",
    "houseNumber": 67,
    "postcode": "N00 BDY",
    "dob": "01/01/1970",
    "policyType": 1
  }
}
```

where `policyType` represents the type of insurance to request a risk score for, and can take the following values:

- 0 - home insurance policy
- 1 - motor insurance policy
- 2 - endowment policy
- 3 - commercial policy
- 4 - public liability insurance policy

The response body will look like this:

```
{
  "insuranceScoreResponse": {
    "timestamp": "2013-05-05T10:46:50.52Z",
    "customerId": 55446511,
    "score": 341
  }
}
```

where `customerId` is an 8-digit number integer and `score` is an integer from 100 to 999.

Based on this information, you create two JSON schemas:

- ▶ One for the request message
- ▶ One for the response

When writing JSON schema you might find it helpful to see the JSON schema specification and related tutorials, which are available on the JSON schema website:

<http://json-schema.org/>

The website also contains a list of validation tools and libraries that might be helpful. Consider validating your schema before running DFHJS2LS. One useful tool is the online JSON schema validator available at the following website:

<http://json-schema-validator.herokuapp.com/>

11.2.1 Writing the request schema

Begin with the request message. Example 11-2 shows a first attempt at a schema that gives the basic structure for the message. This schema expresses the structure of the data, and also states that all of the properties are required, because the documentation does not state any of the fields are optional. If you do not mark properties as required, DFHJS2LS will generate existence flags in the language structure that are set at run time if the fields are present in the transformed JSON.

Example 11-2 Basic schema for the insuranceScoreRequest

```
{
  "type": "object",
  "$schema": "http://json-schema.org/draft-04/schema",
  "required": [
    "insuranceScoreRequest"
  ],
  "properties": {
    "insuranceScoreRequest": {
      "type": "object",
      "properties": {
        "dob": {
          "type": "string"
        },
        "firstName": {
          "type": "string"
        },
        "houseNumber": {
          "type": "string"
        },
        "lastName": {
          "type": "string"
        },
        "policyType": {
          "type": "string"
        },
        "postcode": {
          "type": "string"
        }
      }
    },
    "required": [
      "dob",
      "firstName",
      "houseNumber",
      "lastName",
      "policyType",
      "postcode"
    ]
  }
}
```

The next step is to add some constraints on the sizes of the fields, because otherwise DFHJS2LS assumes the default values, resulting in much padding in the language structure.

For the string fields, use the `minLength` and `maxLength` properties, and for integer fields use `maximum` and `minimum`. You also make some inferences about the lengths of some of these fields.

The resulting schema is shown in Example 11-3. Notice that the `policyType` field has a maximum value of 999 specified, despite the documentation stating the highest acceptable value is 4. This is because a field with a maximum value less than 256 is mapped to a COBOL `PIC X DISPLAY` declaration (because no suitably small binary type is provided in COBOL). The value of 999 maps to a `PIC 999` declaration which is more suitable for your application.

Example 11-3 Improved schema for `insuranceScoreRequest`

```
{
  "type": "object",
  "$schema": "http://json-schema.org/draft-04/schema",
  "required": [
    "insuranceScoreRequest"
  ],
  "properties": {
    "insuranceScoreRequest": {
      "type": "object",
      "properties": {
        "dob": {
          "type": "string",
          "minLength": 10,
          "maxLength": 10
        },
        "firstName": {
          "type": "string",
          "minLength": 1,
          "maxLength": 50
        },
        "houseNumber": {
          "type": "string",
          "minLength": 1,
          "maxLength": 4
        },
        "lastName": {
          "type": "string",
          "minLength": 1,
          "maxLength": 50
        },
        "policyType": {
          "type": "integer",
          "minimum": 0,
          "maximum": 999
        },
        "postcode": {
          "type": "string",
          "minLength": 6,
          "maxLength": 8
        }
      }
    },
    "required": [
      "dob",
      "firstName",
```

```

        "houseNumber",
        "lastName",
        "policyType",
        "postcode"
    ]
}
}
}

```

11.2.2 Writing the response schema

Next, you create a schema for the JSON response message, in a similar way as you did for the request message. Example 11-4 shows the schema. Note the use of the `date-time` value of the `format` property. This indicates that the value is a time stamp in RFC3339 format, which at run time is converted to CICS ABSTIME format.

Example 11-4 JSON schema for insuranceScoreResponse

```

{
  "type": "object",
  "$schema": "http://json-schema.org/draft-04/schema",
  "properties": {
    "insuranceScoreResponse": {
      "type": "object",
      "properties": {
        "customerId": {
          "type": "integer",
          "minimum": 0,
          "maximum": 99999999
        },
        "score": {
          "type": "integer",
          "minimum": 100,
          "maximum": 999
        },
        "timestamp": {
          "type": "string",
          "format": "date-time"
        }
      },
      "required": [
        "customerId",
        "score",
        "timestamp"
      ]
    },
    "required": [
      "insuranceScoreResponse"
    ]
  }
}

```

11.3 Generating the language structures

Now, you have a description of the interface to the service in the form of a JSON schema. Next, generate a language structure to be used by the client application and the necessary artifacts for CICS to transform between JSON and application data.

You do this by running DFHJS2LS, which is the JSON assistant used when starting from a JSON schema. This process is similar to using DFHJS2LS when developing a JSON web service provider, as described in 10.4.2, “Mapping the JSON schema to language structures” on page 117, but you use some different parameters.

Run DFHJS2LS twice, once with the JSON schema for the request, and once with the JSON schema for the response. Each time, a language structure and a CICS bundle containing a JSONTRANSFRM bundle part are generated.

The bundle is generated on zFS, and is later installed into CICS. The CICS TS Feature Pack for Mobile Extensions V1.0 supplies a JCL procedure to start DFHJS2LS in the SDFHMOBI library. The JCL to start it with the request schema is shown in Example 11-5.

Example 11-5 JCL to run DFHJS2LS for the request language structure

```
//JS2LS JOB (MYSYS,AUSER),MSGCLASS=H,
//          CLASS=A,NOTIFY=&SYSUID,REGION=0M
// JCLLIB ORDER='CICS510.SDFHMOBI'
//*
//JS2LS    EXEC DFHJS2LS,USSDIR='cics680',
//          PATHPREF='',JAVADIR='java6_64/J6.0_64'
//INPUT.SYSUT1 DD *
LOGFILE=/u/cicsuser/genapp/json/logs/insuranceScoreRequest.log
PDSLIB=//USER.JS2LS.COPYLIB
PDSMEM=SCREQ
LANG=COBOL
MAPPING-LEVEL=3.0
JSONTRANSFRM=SCOREREQ
BUNDLE=/u/cicsuser/genapp/json/client/insuranceScoreRequest
CHAR-VARYING=NO
JSON-SCHEMA=/u/cicsuser/genapp/json/insuranceScoreRequest.json
*/
```

You must supply the following parameters when starting DFHJS2LS:

LOGFILE	The zFS file where a log of the DFHJS2LS processing is created.
PDSLIB	The partitioned data set where the language structure is created.
PDSMEM	The name of the member in the partitioned data set that is created.
LANG	The high-level language in which the language structure is created.
MAPPING-LEVEL	The level of mapping applied by the JSON assistant. 3.0 or greater can be used, but earlier mapping levels are supported only for compatibility with the SOAP web services assistants.
JSONTRANSFRM	The name of the JSONTRANSFRM bundle part that will be created by CICS when the bundle is installed.
BUNDLE	The zFS location of the bundle that is created.

CHAR-VARYING=NO Suppresses the generation of length fields for variable-length string values.

JSON-SCHEMA The zFS location of the JSON schema used as input.

Full details of all the parameters for DFHJS2LS is found in the “DFHJS2LS: JSON schema to high-level language conversion for linkable interface” topic of the CICS TS Feature Pack for Mobile Extensions Information Center. The following website is for CICS TS 5.1:

http://pic.dhe.ibm.com/infocenter/cicsts/v5r1/topic/com.ibm.cics.ts.mobileextensions.doc/reference/dfhws_js2lsapi.html

Example 11-6 shows the JCL for the response structure. This is similar to the JCL for the request schema, except for the values of the **LOGFILE**, **PDSMEM**, **JSONTRANSFRM**, and **BUNDLE** parameters.

Example 11-6 JCL to run DFHJS2LS for the response language structure

```
//JS2LS JOB (MYSYS,AUSER),MSGCLASS=H,
//          CLASS=A,NOTIFY=&SYSUID,REGION=0M
// JCLLIB ORDER='CTS.CICS510.SDFHMOBI'/*
/* The following line is changed by APAR PK04055 @BA04055
//JS2LS EXEC DFHJS2LS,USSDIR='cics680',
//          PATHPREF='',JAVADIR='java6_64/J6.0_64'
//INPUT.SYSUT1 DD *
LOGFILE=/u/cicsuser/genapp/json/logs/insuranceScoreResponse.log
PDSLIB=//USER.JS2LS.COPYLIB
PDSMEM=SCRESP
LANG=COBOL
MAPPING-LEVEL=3.0
JSONTRANSFRM=SCORERESP
BUNDLE=/u/cicsuser/genapp/json/client/insuranceScoreResponse
CHAR-VARYING=NO
JSON-SCHEMA=/u/cicsuser/genapp/json/insuranceScoreResponse.json
*/
```

When running DFHJS2LS, the following error message can occur:

```
DFHPI9523E An unexpected error occurred whilst processing file
"//USER.JS2LS.COPYLIB(CRREQ01)". The problem is: "//USER.JS2LS.COPYLIB(CRREQ01)".
```

This normally indicates that the partitioned data set (PDS) cannot be opened for output because a user had a member of the PDS open for editing in Interactive System Productivity Facility (ISPF) or IBM Rational Developer for IBM System z.

Also, if you run DFHJS2LS more than one time, with the same value of **BUNDLE** parameter, the following message can occur:

```
DFHPI9683W Bundle directory "/u/cicsuser/genapp/client/insuranceScoreRequest"
already exists and may contain files that are inconsistent with the new Bundle
manifest file.
```

This message can be safely ignored *if* you have maintained the same value of the **JSONTRANSFRM** parameter and are rerunning the assistant due to a change in the input schema or mapping parameters. However, if a different bundle already exists at this location, you must choose a different one or delete the existing directory first.

11.4 Defining the CICS resources

The next step is to define the resources used by CICS. These resources are named in the application, and are required to test it, so they must be defined before the program can be developed. You define BUNDLE resources for each **JSONTRANSFRM** and a URIMAP resource using the CICS Explorer.

11.4.1 Defining the BUNDLE resources

When you run DFHJS2LS to create the language structures (see 11.3, “Generating the language structures” on page 135), it also creates a bundle directory on zFS. This contains a JSBIND file and a **JSONTRANSFRM** bundle part, which CICS uses to perform the transformation between application data and JSON. You must create and install a CICS BUNDLE resource for both the request and response bundles.

Follow these steps to create and install the BUNDLE resources in CICS Explorer:

1. Click **File** → **New** → **Other**.
2. In the New window, expand CICS Definitions and select **Bundle Definition**, as shown in Figure 11-2. Click **Next**.

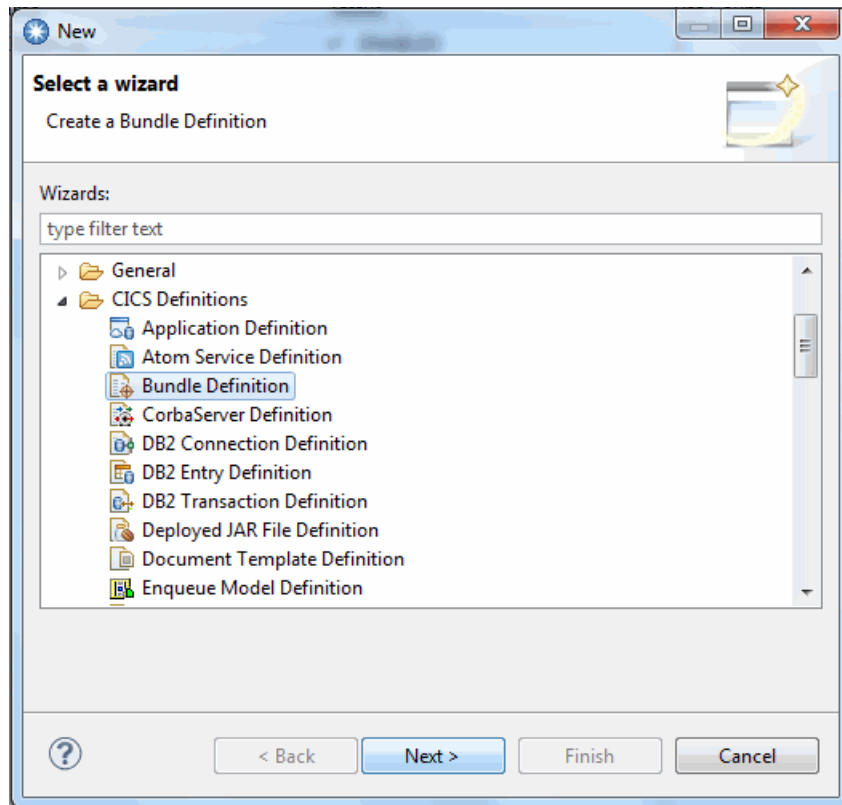


Figure 11-2 Creating a new bundle definition

3. On the Create Bundle Definition page, complete these steps:
 - a. Enter the name of a CICSplex where the definition will be created in the CICSplex field.
 - b. If you want to create the resource in a CICS system definition data set (CSD), for example if you are connected to a stand-alone CICS region, select the **Region (CSD)** check box and enter the name of the region in the adjacent field.

- c. Enter the name of the CSD or resource group where the bundle definition will be created in the Resource/CSD Group field.
- d. Enter the name of the bundle for the request transform in the Name field.
- e. Click **Browse** and choose the bundle directory on zFS that was created by DFHJS2LS when processing the request schema.

The completed page is shown in Figure 11-3.

Figure 11-3 Specifying the attributes of a new bundle definition

4. Click **Finish**.
5. Repeat steps 1-4, entering details for the response transform.
6. Select **Definitions** → **Bundles Definitions** to show the Bundle Definitions view.
7. In the CICSplex Repositories view, select the group that you specified in step 3c.

- Select both of the bundle definitions listed in the Bundle Definitions view, as shown in Figure 11-4.

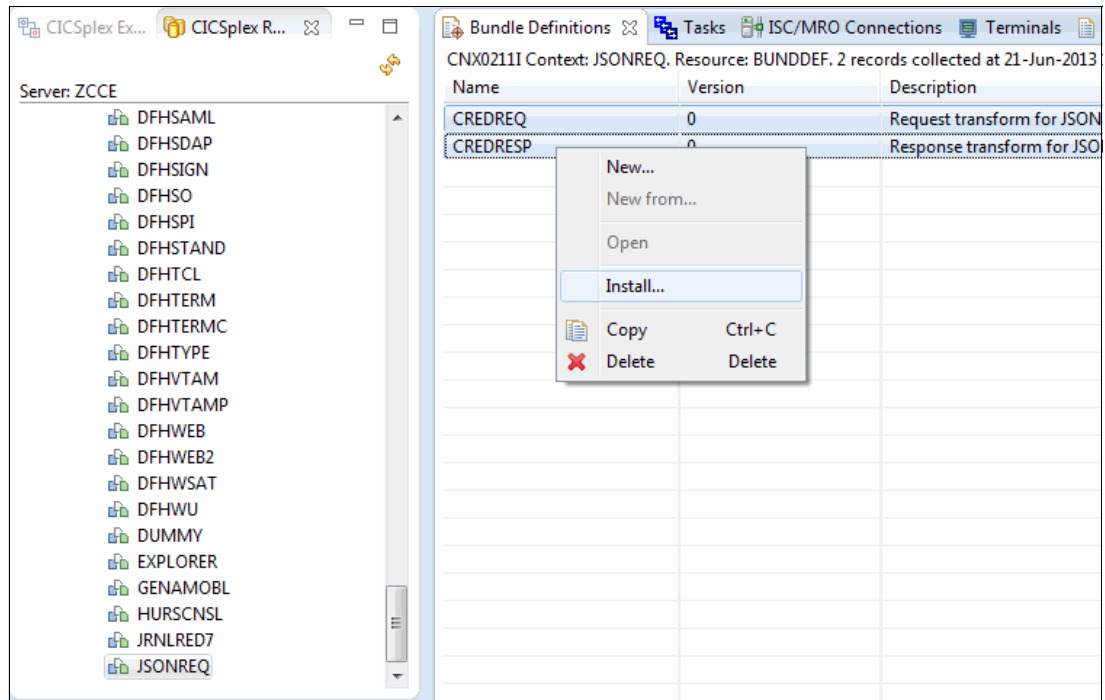


Figure 11-4 Locating the bundle resources to install

- Right-click the selected definitions and click **Install**.
- In the Perform Operation window, select the region or system group into which you want to install the definitions, and then click **OK**.

You can now view the BUNDLES and BUNDLEPARTs to verify they installed correctly. To view the BUNDLES in CICS Explorer, click **Operations** → **Bundles**. To view the corresponding BUNDLEPARTs, right-click a BUNDLE and select **Show Bundle Parts**. You can see a single **JSONTRANSFRM** BUNDLEPART for each BUNDLE, as shown in Figure 11-5.

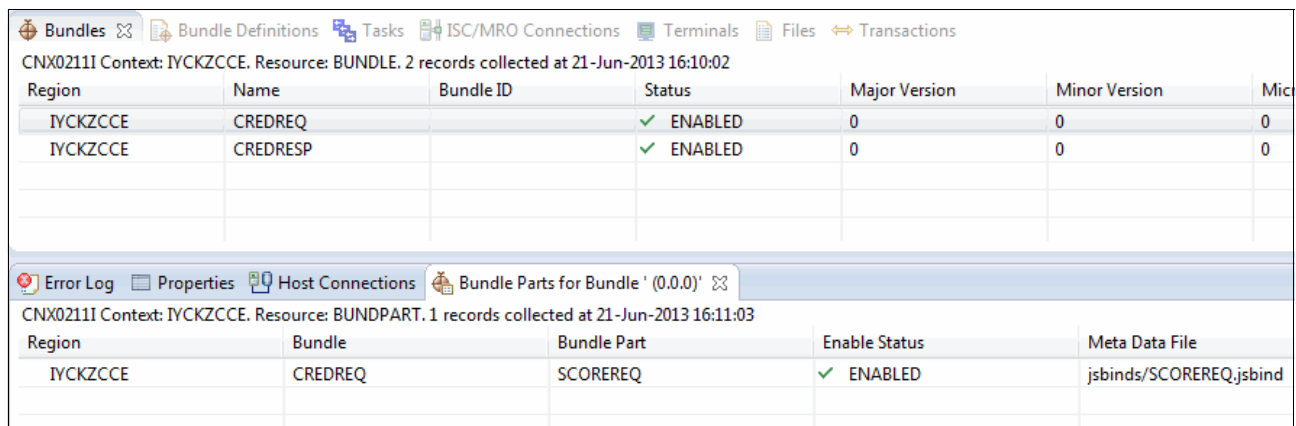


Figure 11-5 Viewing the Bundle Parts for the credreq bundle

11.4.2 Defining the URIMAP resource

You can define a URIMAP resource that specifies the URI of the JSON service that you are writing a client application for. This is not required (you can specify all the parameters on the **WEB OPEN** and **WEB CONVERSE** commands instead). However, creating a URIMAP resource has the following advantages:

- ▶ You can use an SSL client certificate to authenticate with the server.
- ▶ You can avoid coding the URI in your application, so that it can be updated by simply modifying the URIMAP definition.
- ▶ You can enable connection pooling, so that all of the connections to the same host share a single HTTP connection.

To define and install a URIMAP resource in CICS Explorer, follow these steps:

1. Click **File** → **New** → **Other**.
2. In the New window, expand **CICS** Definitions, and **URI Map Definition**. Click **Next**.
3. On the Create URI Map Definition page, complete these steps:
 - a. Enter the name of a CICSplex where the definition will be created in the CICSplex field.
 - b. If you want to create the resource in a CSD (for example if you are connected to a stand-alone CICS region), select the **Region (CSD)** check box and enter the name of the region in the adjacent field.
 - c. Enter the name of the CSD or resource group where the URIMAP definition will be created in the Resource/CSD Group field.
 - d. Enter the name of the URIMAP in the Name field.
 - e. Enter the host where the JSON web services is located in the Host field.
 - f. Enter the path to the service in the Path field.
 - g. Click **Client**, and enter the TCP/IP port for the service in the Port field.

The completed page is shown in Figure 11-6 on page 141.

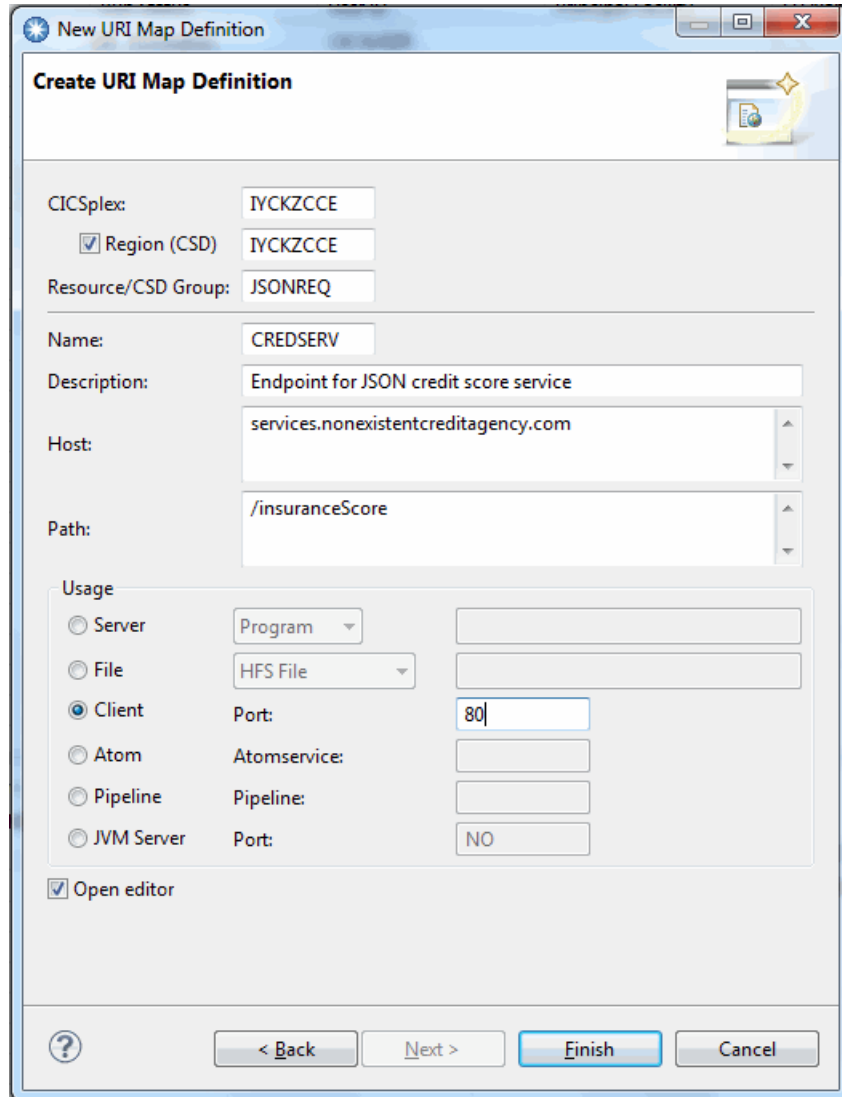


Figure 11-6 Specifying the attributes of the URIMAP

4. Click **Finish**.
5. In the URI Map Definition editor, follow these steps:
 - a. If you want connections that were opened using this URIMAP resource to be pooled for reuse, specify the `SOCKETCLOSE` attribute as the length of time for which CICS keeps the connection in the pool after the application program has finished using it. See the “Connection pooling for HTTP client performance” topic in the CICS Information Center for information about how CICS manages pooled connections, and how connection pooling improves application performance.
For CICS TS 5.1, this is found at the following website:
http://pic.dhe.ibm.com/infocenter/cicsts/v5r1/topic/com.ibm.cics.ts.internet.doc/topics/dfht3_connpool.html
 - b. To configure security for the connection to the server, see Chapter 7, “Security and workload management” on page 69.
 - c. Press **Ctrl + S** to save your changes.

- d. From the resource drop-down menu, select **Install**.
- e. In the Perform Operation window, select the region or system group that you want to install the definitions into, and then click **OK**.

Figure 11-7 illustrates the URI Map Definition editor.

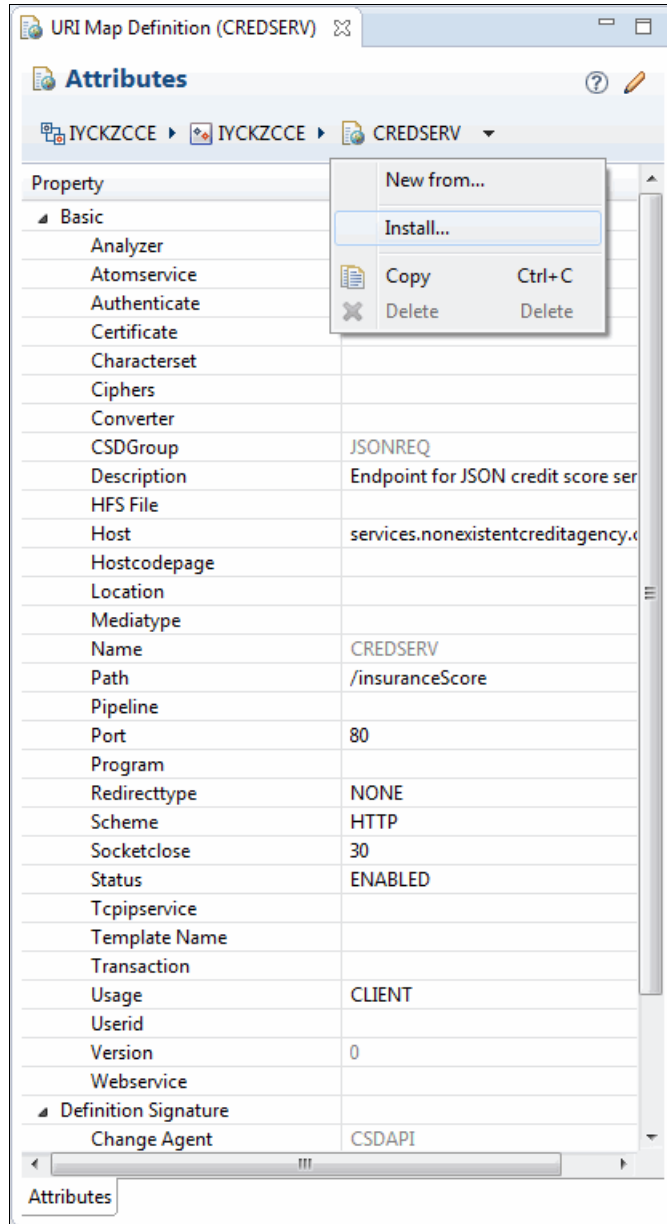


Figure 11-7 Editing the attributes of the URIMAP

11.5 Developing the application program

The final task is to write the application program that will call the remote service. A complete sample COBOL program is provided, and this section provides information about each part of it.

11.5.1 Transforming the request data

The first task that your application program might need to perform is to generate the JSON request message. Whether you need to do this depends on the service you are calling. Some JSON web services can take input from the URI, either from the path or the query string, rather than the request body.

If the service is called using the HTTP **GET** method, a request body cannot be provided. In the scenario presented in this chapter, the service uses a request-response pattern using the **POST** method, so both the request and response are contained in the HTTP body. If you do not need to generate a JSON request, you can skip to 11.5.2, “Sending the request” on page 145. You can use the linkable interface to transform application data to JSON for the request message. You must first set up the containers as noted in the following steps:

1. Put the name of the JSONTRANSFRM bundle part for the request (as specified by the **JSONTRANSFRM** parameter on DFHJS2LS) in the DFHJSON-TRANSFRM container.
2. If you want to use a JVM server other than DFH\$AXIS, put the name of the JVMSERVER resource in the DFHJSON-JVMSEVR container.
3. Put the application data that you want to transform in the DFHJSON-DATA container.
4. Then perform a **LINK PROGRAM** to DFHJSON, passing the channel where you have put the containers.

Example 11-7 is an excerpt from the sample COBOL program that performs these tasks.

Example 11-7 Sample COBOL to transform the request message

```
MOVE 'JOE' TO FIRSTNAME
MOVE 'BLOGGS' TO LASTNAME
MOVE 67 TO HOUSENUMBER
MOVE '10/10/1984' TO DOB
MOVE 'N00 BDY' TO POSTCODE
MOVE 3 TO POLICYTYPE

EXEC CICS PUT CONTAINER('DFHJSON-TRANSFRM')
CHANNEL('CHAN')
FROM('SCOREREQ')
CHAR
RESP(CICS-RESP)
END-EXEC
PERFORM CHECK-RESP

EXEC CICS PUT CONTAINER('DFHJSON-DATA')
CHANNEL('CHAN')
FROM(REQUEST-DATA)
RESP(CICS-RESP)
END-EXEC
PERFORM CHECK-RESP

*
Link to the transformer
EXEC CICS LINK PROGRAM('DFHJSON')
CHANNEL('CHAN')
RESP(CICS-RESP)

END-EXEC
PERFORM CHECK-RESP
```

Handling Errors

If an error occurs during transformation, CICS puts an error code in the DFHJSON-ERROR container and returns to the application. Under some circumstances, CICS also puts further information about the error in the DFHJSON-ERRORMSG container. After linking to DFHJSON, you can check for the presence of the DFHJSON-ERROR container and take action accordingly.

Some types of errors indicate a configuration error, such as the JSONTRANSFRM resource not being defined or enabled. Other types of errors indicate a problem with the data transformation, such as invalid JSON or a mismatch between the type of data provided and the data that was expected. In these situations, it can be helpful to capture the contents of the DFHJSON-ERRORMSG container.

Example 11-8 shows a COBOL procedure that can be started after linking to DFHJSON. It checks for the DFHJSON-ERROR container, and (if the error container is present) displays the error code on the terminal. If the DFHJSON-ERRORMSG container is present, the first 256 bytes of its contents are sent to the transient data queue (TDQ) CESE using a COBOL **DISPLAY** statement.

Example 11-8 Sample COBOL routine for handling errors return by the linkable interface

```
HANDLE-ERROR.
      EXEC CICS GET CONTAINER('DFHJSON-ERROR') CHANNEL('CHAN')
              INTO(TRANS-RESP)
              RESP(CICS-RESP)
              END-EXEC
      IF CICS-RESP EQUAL DFHRESP(NORMAL)
*      Error container is present, output value
      MOVE TRANS-RESP TO ERROR-DISPLAY
      EXEC CICS SEND TEXT FROM(BAD-RESP-MSG)
              ERASE END-EXEC

      MOVE 256 TO ERROR-LENGTH

      EXEC CICS GET CONTAINER('DFHJSON-ERRORMSG')
              CHANNEL('CHAN')
              INTO(ERROR-MSG)
              RESP(CICS-RESP)
              FLENGTH(ERROR-LENGTH)
              END-EXEC

      IF CICS-RESP EQUAL DFHRESP(NORMAL)
      DISPLAY ERROR-MSG
      END-IF
      EXEC CICS RETURN END-EXEC
END-IF
EXIT.
```

Full details of the possible errors can be found in the “DFHJSON-ERROR container” topic in the CICS TS Feature Pack for Mobile Extensions Information Center. For CICS TS 5.1 this information can be found at the following website:

http://pic.dhe.ibm.com/infocenter/cicsts/v5r1/topic/com.ibm.cics.ts.mobileextensions.doc/reference/dfhjson_error.html

11.5.2 Sending the request

The next step is to send the request to the JSON web service over HTTP. This is accomplished using **WEB** commands.

Opening the connection

First, open a connection using a **WEB OPEN** command. This opens an HTTP connection, or reuses an existing one (if connection pooling is enabled). If you use a URIMAP resource as described in 11.4.2, “Defining the URIMAP resource” on page 140, you can name the resource on the **WEB OPEN** command. Otherwise, you must code the **HOST** and **PORTNUMBER** parameters to specify the server to connect to.

A **WEB OPEN** command using a URIMAP is shown in Example 11-9. The **SESSTOKEN** specifies a data area into which CICS will put a session token. This must be specified on all subsequent **WEB** commands to identify the connection.

Example 11-9 Opening the connection to the remote service using a URIMAP

```
EXEC CICS WEB OPEN
                URIMAP('CREDSERV')
                RESP(CICS-RESP)
                RESP2(CICS-RESP2)
                SESSTOKEN(TOKEN) END-EXEC
```

Sending the data and receiving the response

Now you are ready to make the request to the JSON web service. You will use a **WEB CONVERSE** command to send the request and receive the response together. You can also use separate **WEB SEND** and **WEB RECEIVE** commands.

The parameters that you specify on the command depend on the interface to the service that you are calling. For example, if your service takes input from the query string, you must specify the **QUERYSTRING** parameter, but if your service expects a JSON body, you must specify the **FROM** or **CONTAINER** parameters.

Specifying the path to the service

Example 11-10 on page 146 uses the **URIMAP** parameter. CICS then uses the **PATH** attribute of the corresponding URIMAP to obtain the path to the JSON web service. Alternatively, you can use the **PATH** and optionally **PATHLENGTH** parameters to specify the path directly. If your service requires a query string, specify the **QUERYSTRING** parameter and optionally the **QUERYSTRINGLEN** parameter.

Specifying the HTTP method and media type

You must specify the HTTP method used to call the JSON web service. You can either specify it directly on the command (as shown in Example 11-10 on page 146), or use the **METHOD** parameter. You must also specify the **MEDIATYPE** parameter to indicate the type of data that you are sending. For JSON, the usual value is `application/json`, but other values can be supported, so you can check with the operator of the JSON web service as to what they expect.

Note: The data area specified on the **MEDIATYPE** parameter must be 56 bytes in length, so you must add trailing spaces to the value.

Specifying the request and response data

If your service expects JSON in the request body, you must code either the **CONTAINER** or **FROM** parameters. If you use the **CONTAINER** parameter, you can pass the DFHJSON-JSON container returned by the linkable interface, as shown in Example 11-10.

If your service will provide JSON data in the response that you want to transform, you can also specify DFHJSON-JSON on the **TOCONTAINER** parameter so that the response data can be passed directly to the linkable interface. If using the **CONTAINER** or **TOCONTAINER** parameters, you can specify the channel using the corresponding **CHANNEL** or **TOCHANNEL** parameters, or CICS uses the current channel.

Additionally, you can specify the **STATUSCODE** parameter. This is a data area in which CICS places the HTTP response code. Your application can check this value to determine if the operation was completed successfully, or if an error occurred.

Example 11-10 shows the complete **WEB CONVERSE** command.

Example 11-10 WEB CONVERSE command to communicate with the insurance score web service

```
EXEC CICS WEB CONVERSE
                                URIMAP('CREDSERV') POST
                                CONTAINER('DFHJSON-JSON')
                                CHANNEL('CHAN')
                                MEDIATYPE(CONTENT-TYPE)
                                TOCONTAINER('DFHJSON-JSON')
                                TOCHANNEL('CHAN')
                                STATUSCODE(HTTP-RESP)
                                STATUSTEXT(HTTPSTATUS)
                                RESP(CICS-RESP)
                                RESP2(CICS-RESP2)
                                SESSTOKEN(TOKEN) END-EXEC
```

Tidying up

After the **WEB CONVERSE**, or the final **WEB SEND** or **WEB RECEIVE** command, you can issue a **WEB CLOSE** command. This signals to CICS that the application has finished using the HTTP connection. If you use connection pooling, the HTTP connection might not be closed, but instead returned to the pool for reuse. Example 11-11 shows the **WEB CLOSE** command.

Example 11-11 WEB CLOSE command to indicate the end of the session

```
EXEC CICS WEB CLOSE SESSTOKEN(TOKEN) END-EXEC
```

11.5.3 Transforming the response body

If the JSON web service that you have called provides JSON in the response body, you can transform this to application data for further processing. This can be accomplished by using the linkable interface in a similar way to that described in 11.5.1, “Transforming the request data” on page 143.

Some services indicate the success or failure of the operation simply using the HTTP response code, in which case this step is not required. This section explains how the example program transforms the response from the insurance score service into application data.

The steps to use the linkable interface to transform JSON to application data are similar to those used to transform application to JSON, except that some of the containers differ. Before calling the transformer, set up the containers using the following points:

- ▶ Put the name of the JSONTRANSFRM for the response (as specified by the **JSONTRANSFRM** parameter on DFHJS2LS) in the DFHJSON-TRANSFRM container.
- ▶ If you want to use a JVM server other than DFH\$AXIS, put the name of the JVMSERVER resource in the DFHJSON-JVMSEVR container.
- ▶ Put the JSON that you want to transform in the DFHJSON-JSON container.

Note: If you have previously used the linkable interface to transform application data to JSON, you must either use a **DELETE CONTAINER** command to delete the DFHJSON-DATA container before calling DFHJSON, or use a different channel. Otherwise, both containers will be present on the channel and you will receive error code 14.

Then, use a **LINK PROGRAM** command to call DFHJSON. If the transformation occurs successfully, CICS puts the application data corresponding to the JSON that you provided in the DFHJSON-DATA container. If an error occurs, CICS puts an error code in the DFHJSON-ERROR container, see “Handling Errors” on page 144 for more information. Example 11-12 demonstrates how to set up the containers and call the linkable interface to transform JSON.

Example 11-12 Calling linkable interface to transform the JSON response from insurance score service

```

EXEC CICS DELETE CONTAINER('DFHJSON-DATA')
          CHANNEL('CHAN')
          END-EXEC

          EXEC CICS PUT CONTAINER('DFHJSON-TRANSFRM')
          CHANNEL('CHAN')
          FROM('SCORERESP')
          RESP(CICS-RESP)
          CHAR
          END-EXEC
          PERFORM CHECK-RESP

*      Link to the transformer
EXEC CICS LINK PROGRAM('DFHJSON')
          CHANNEL('CHAN')
          RESP(CICS-RESP)
          END-EXEC
          PERFORM CHECK-RESP

          PERFORM HANDLE-ERROR

          EXEC CICS GET CONTAINER('DFHJSON-DATA') CHANNEL('CHAN')
          INTO(RESPONSE-DATA)
          RESP(CICS-RESP)
          END-EXEC.
          PERFORM CHECK-RESP

```

11.6 Testing the sample application

If you want to test the sample application, you will need to set up the client application and the provider service. You can follow the steps in the preceding sections to create the client application, or you can use the materials supplied with this book. For information about how to obtain these, see Appendix C, “Additional material” on page 175. In either case, you must perform the following tasks sequentially:

1. Follow the steps in 5.2.2, “How to configure CICS as a service provider” on page 34 to configure your system for JSON web services.
2. Copy the `credit.wsbind` file provided with this book to the `pickup` directory of the PIPELINE you created in “Defining and installing a PIPELINE” on page 40, and perform a PIPELINE scan.
3. Create the resource definitions for the requester application as described in 11.4, “Defining the CICS resources” on page 137 if you have not done so already. You can use the bundles supplied with the book if you do not want to create them yourself.
4. Compile the sample programs CREDIT and REQUEST and put them in a load library that is part of the DFHRPL concatenation. Alternatively, create and install a LIBRARY definition that references the load library.
5. If you do not use program autoinstall, create and install PROGRAM definitions for CREDIT and REQUEST.
6. Create and install a TRANSACTION definition that calls REQUEST.

You can then start the transaction from a terminal. If it completes successfully, you can see a message on the terminal indicating the insurance score returned from the service.



IBM Worklight for CICS

This chapter describes how to call a Customer Information Control System (CICS) JavaScript Object Notation (JSON) service hosted in CICS setup using IBM Worklight's adapter and IBM Worklight's client JavaScript application programming interface (API), where the adapter and API run on the mobile device itself.

This chapter includes the following topics:

- ▶ “Creating a Worklight adapter” on page 150
- ▶ “Testing the Worklight adapter” on page 154
- ▶ “Calling the Worklight adapter from the Worklight client code” on page 156

12.1 Creating a Worklight adapter

This process assumes that a blank empty Worklight project was created before creating a Worklight adapter.

To create a Worklight adapter, perform the following steps:

1. Right-click the adapters folder and select **New** → **Worklight Adapter**. See Figure 12-1.

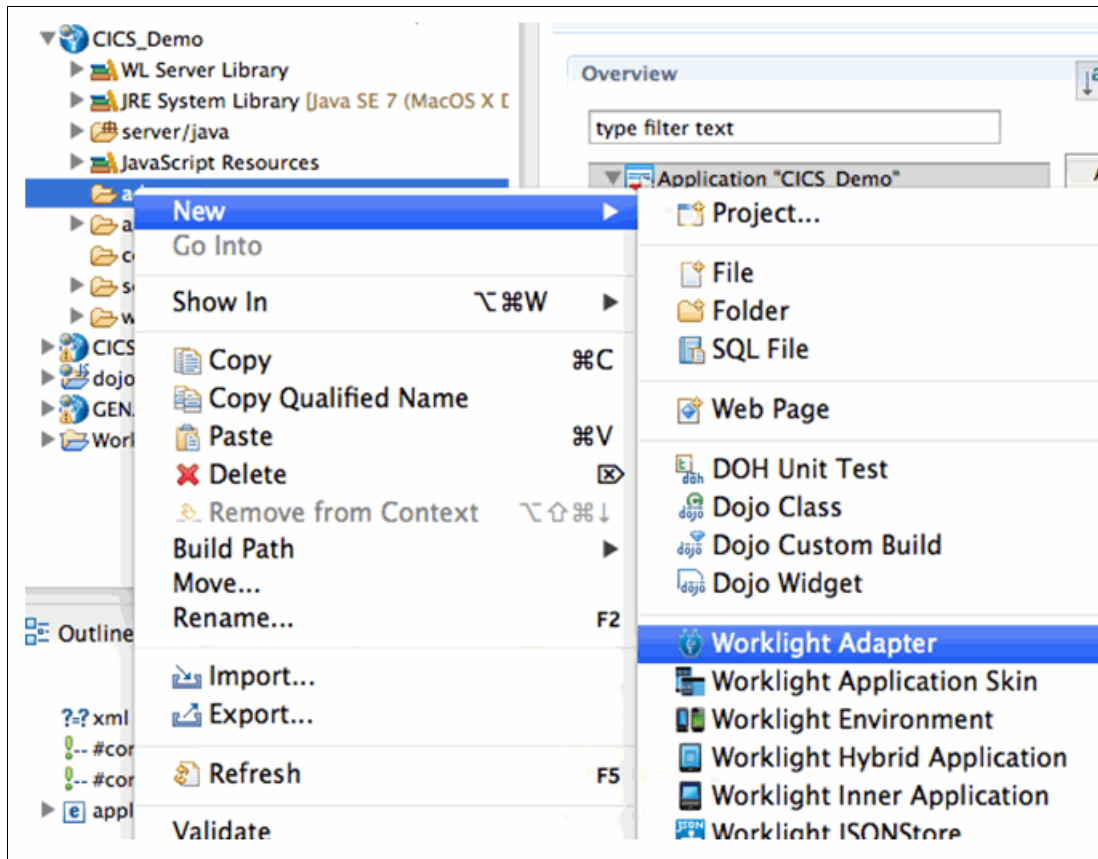


Figure 12-1 Worklight adapter selection

2. Select the Hypertext Transfer Protocol (HTTP) adapter as the type, because you are calling an HTTP JSON Service inside CICS. Give the adapter a name and select **Finish**. See Figure 12-2 on page 151.

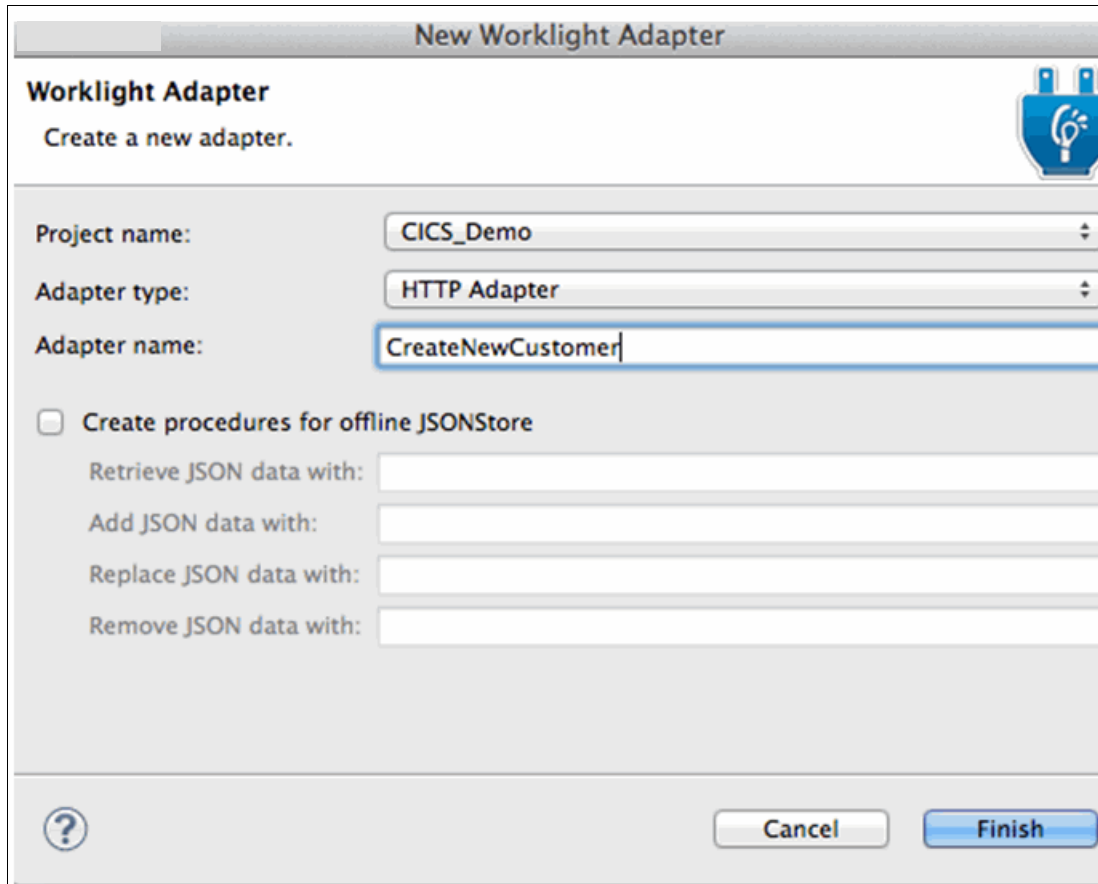


Figure 12-2 Create a new adapter

3. The adapter rich page editor opens, providing for entry of the Domain and port for the service. Click **Connectivity** → **Connection Policy** to enter that data. See Figure 12-3.

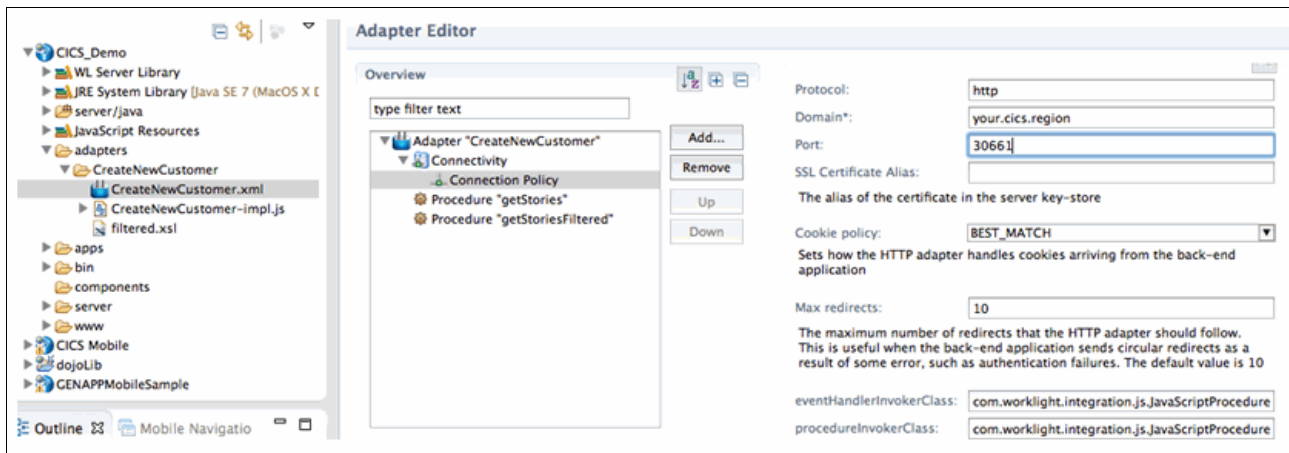


Figure 12-3 Connection Policy

4. Use either the Basic HTTP Authentication or Secure Sockets Layer (SSL), following the guidelines in 7.3, “Worklight security configuration” on page 78.

- Click **Procedure "getStories"**, (shown in Figure 12-3 on page 151), then alter the name to match the operation of your service. In this example, the procedure is named `addNewCustomer` (as shown in Figure 12-4).

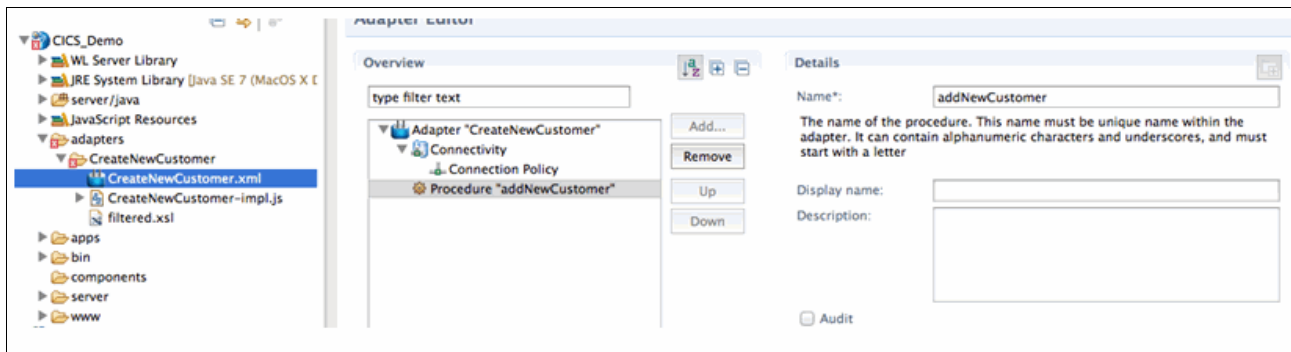


Figure 12-4 Procedure editing

- Select **Procedure "getStoriesFiltered"** (shown in Figure 12-3 on page 151) and click **Remove**. This will remove the procedure seen in Figure 12-6 on page 153. You will notice there is a red cross on the adapter folder at this point, because the Extensible Markup Language (XML) configuration file for the adapter does not match the implementation `.js` file. Remove the `filtered.xml` file, because it is not needed in this example.
- Open the `CreateNewCustomer-impl.js` file and delete the `getStoriesFiltered` function.
- Rename `getStories` to be `addNewCustomer`.

The `CreateNewCustomer-impl.js` should now look similar to that shown in Figure 12-5.

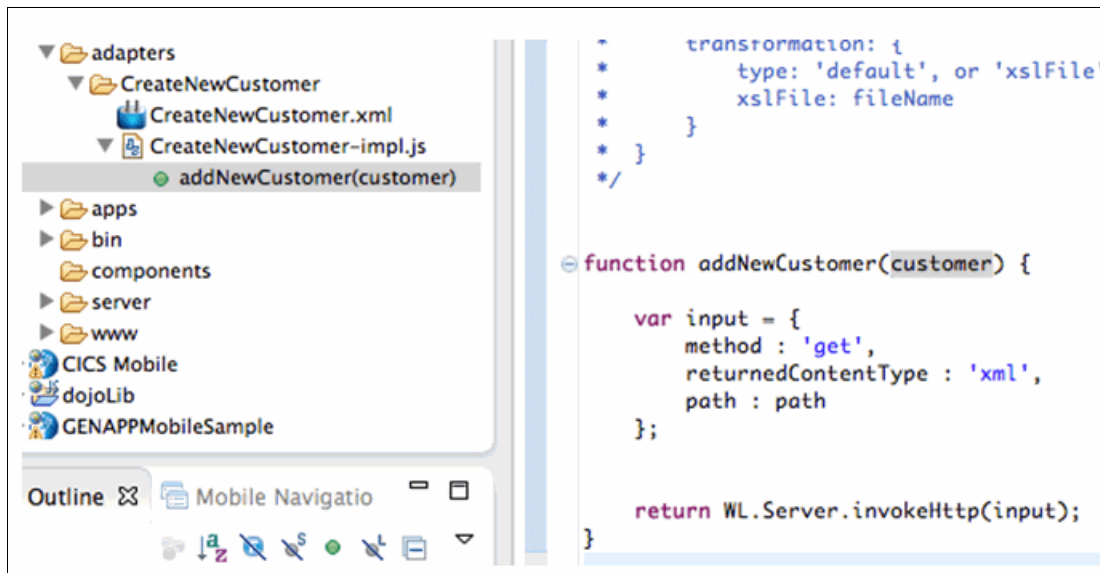


Figure 12-5 The `addNewCustomer(customer)` function

- Now, add in some parameters to the `addNewCustomer` JavaScript function. The service requires a first name, last name, date of birth, ZIP code, cell number, and email address.

These parameters are then added into a JSON payload object, similar to Example 12-1.

Example 12-1 JSON web service payload

```
{
  "cust_details": {
    "first_name": "James", "last_name": "Smith",
    "date_of_birth": "2001-01-01",
    "zipcode": "S0212JN",
    "cell_number": "07756576667",
    "email_address": "james.smith@anycompany.com"
  }
}
```

10. The request then needs to be posted to the CICS service that tells Worklight to expect a plain response payload. With this information, Worklight knows that a conversion of the Representational State Transfer (REST) response, which in this case is a line of text, needs to be converted back to a JSON object. See Figure 12-6.

```
function addNewCustomer(firstName,lastName,dob,zipCode,cellNo,emailAddress) {
  var pathURL = "/genapp/CustService/";
  var request=
  {
    "cust_details":{
      "first_name" : firstName,
      "last_name" : lastName,
      "date_of_birth" : dob,
      "zipcode" : zipCode,
      "cell_number" : cellNo,
      "email_address" : emailAddress
    }
  };

  var input = {
    method : 'post',
    returnedContentType : 'plain',
    path : pathURL,
    body : {
      content: JSON.stringify(request),
      contentType: 'application/json; charset=utf-8'
    }
  };
  return WL.Server.invokeHttp(input);
}
```

Figure 12-6 Completed addNewCustomer function

12.2 Testing the Worklight adapter

The next step is to test the new Worklight adapter. Before continuing, it is important to ensure that the CICS service is installed and enabled correctly. Ensure also that the domain and port combinations, in the adapter's XML configuration file going to the service's hosting location, are accurate.

To test the service of the Worklight adapter, use the following steps:

1. Right-click the adapter folder **CreateNewCustomer** and select **Run As** → **3 Invoke Worklight Procedure**. See Figure 12-7.

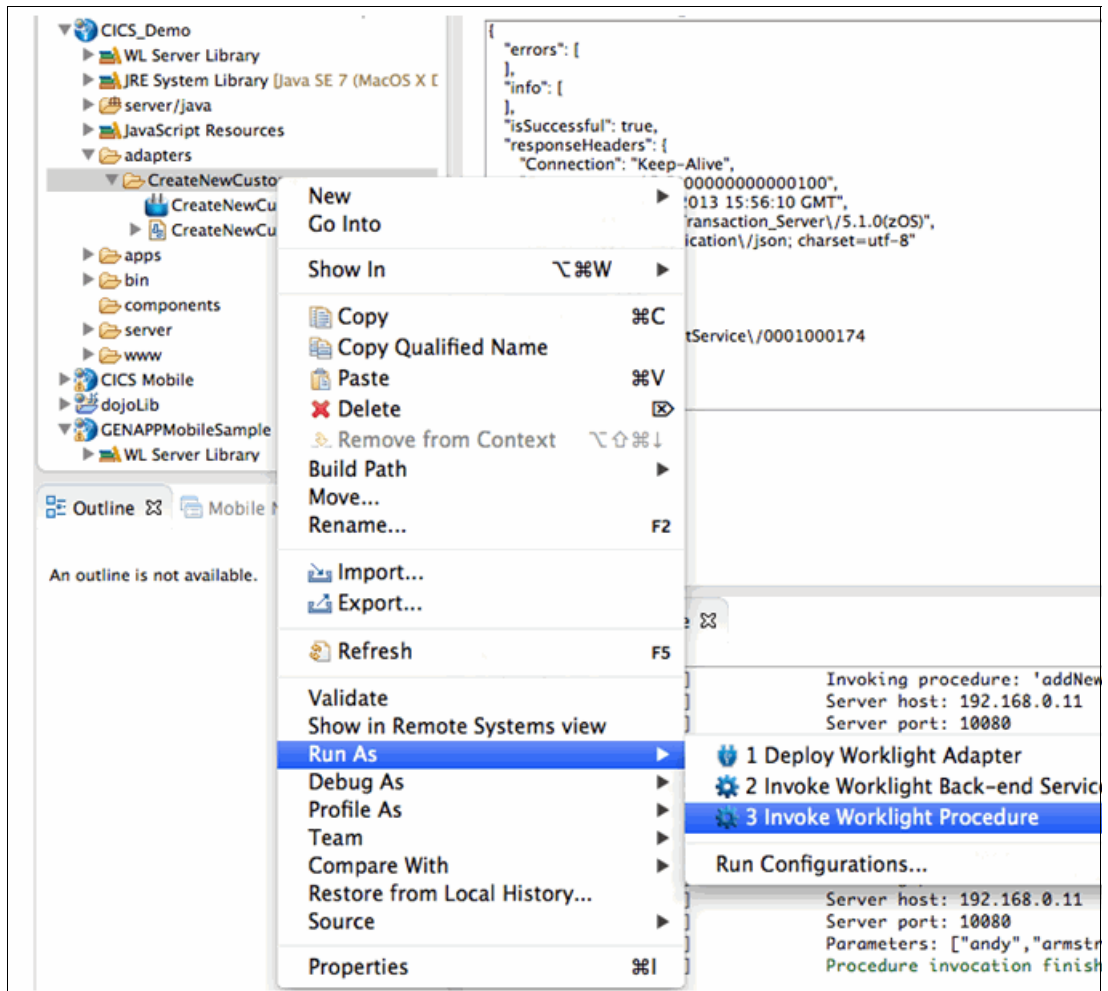


Figure 12-7 The Invoke Worklight procedure

2. This brings up a dialog asking for the procedure name to start, and the list of parameters to be passed to the adapter. At this point, select **addNewCustomer** from the Procedure name drop-down list.

3. Enter parameters (each enclosed within quotation marks because they are string variables). Next, click **Run**. See Figure 12-8.

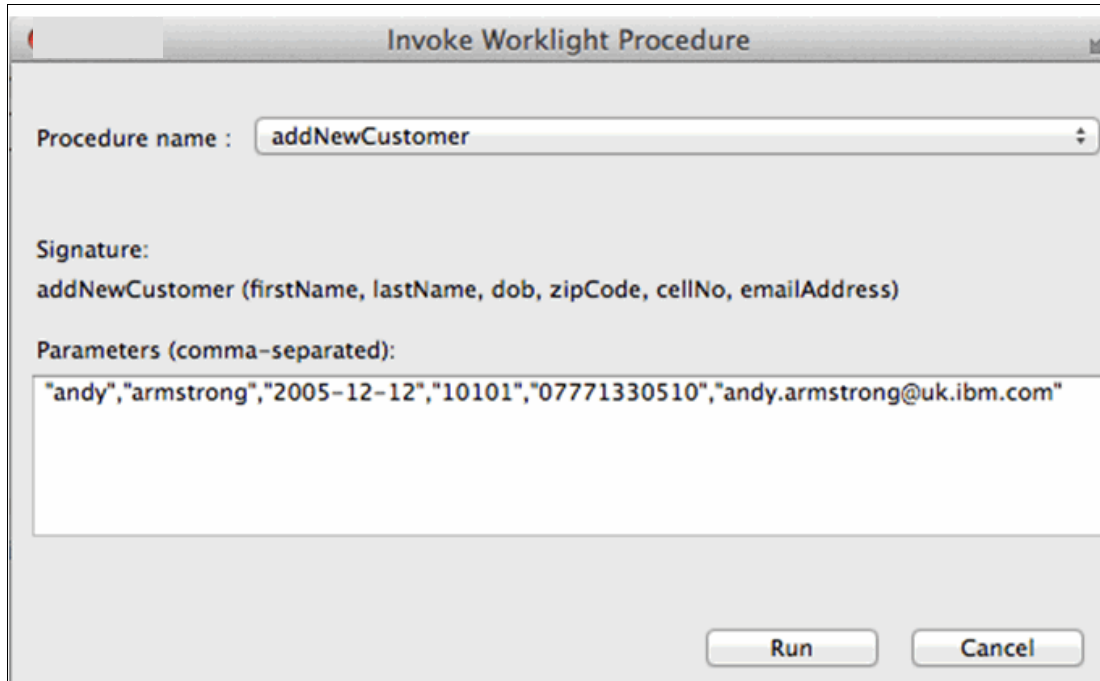


Figure 12-8 Start Worklight procedure parameters

4. Worklight will then start the back-end service and serve the response within a browser window (within the Eclipse view by default). See Figure 12-9.

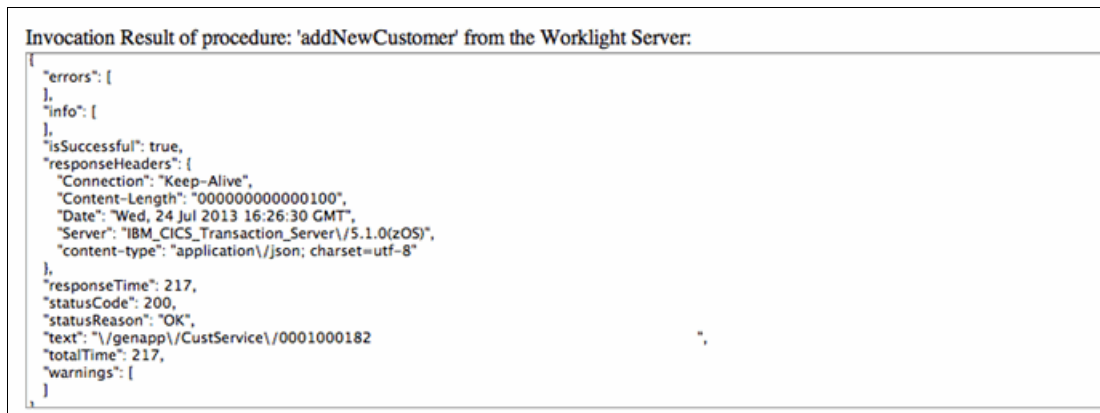


Figure 12-9 Back-end service response

The plain response from CICS was converted into a JSON object, with the text attribute being the Uniform Resource Locator (URL) for the newly created customer.

12.3 Calling the Worklight adapter from the Worklight client code

For the Worklight adapters to be useful, they have to be started from the Client application. Go to the apps/CICS_Demo/common/js folder, and you will notice that the CICS_Demo.js file was created. This is a JavaScript file that is imported by the CICS_Demo.html, which is our main client application starting point.

To import this, perform the following steps:

1. Open CICS_Demo.js and use the example in Figure 12-10 to write the adapter invocation call function.

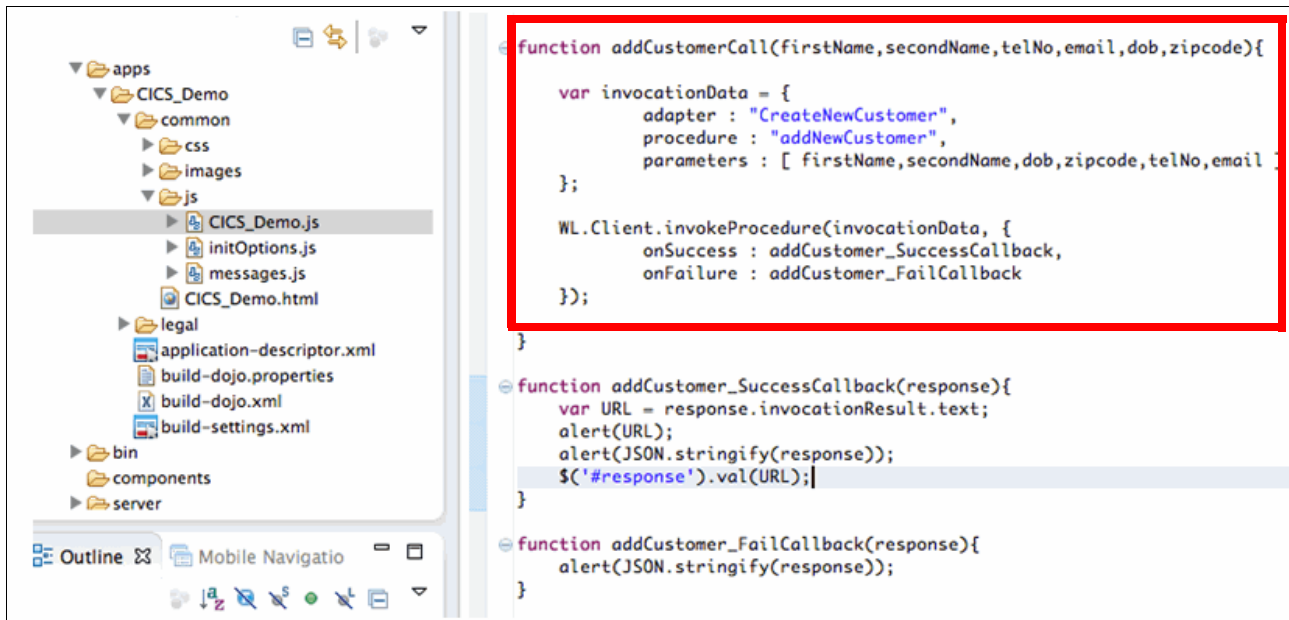


Figure 12-10 Adapter invocation call function

There are two callbacks used when starting the procedure:

- The onSuccess function is called if the procedure call succeeded.
- The onFailure function is called if the procedure call failed.

2. The next step is to create a form on the CICS_Demo.html page, which starts this new JavaScript function to call the REST-conforming (RESTful) service hosted in CICS, to add a new customer. Open the CICS_Demo.html file and alter it to contain a set of fields for the input, a button to start the Worklight adapter, and a field for the URL response from CICS. See Figure 12-11 on page 157.

```

content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no" /
<meta name="apple-mobile-web-app-capable" content="yes">
<link rel="shortcut icon" href="images/favicon.png">
  <link rel="apple-touch-icon" href="images/apple-touch-icon.png">
  <link rel="stylesheet" href="css/CICS_Demo.css">
  <script>window.$ = window.jQuery = WLJQ;</script>
  <script type="text/javascript" src="dojox/mobile/deviceTheme.js"></script>
  <script type="text/javascript" data-dojo-config="isDebug: false, async: true, parseOnLoad: true, mb
</head>
<body id="content" style="display: none;">
  <div data-dojo-type="dojox.mobile.ScrollableView" id="scroll"
    data-dojo-props="selected:true,scrollDir:'v'">
    <div data-dojo-type="dojox.mobile.Heading"
      data-dojo-props="label:'CICS Demo',moveTo:'view0',fixed:'top'"></div>
    <div data-dojo-type="dojox.mobile.ScrollableView" id="view0" data-dojo-props="selected:true">
      <input data-dojo-type="dojox.mobile.TextBox" placeholder="First Name" id="firstName">
      <br/>
      <br/><input data-dojo-type="dojox.mobile.TextBox" placeholder="Last Name" id="lastName">
    <br/>
    <br/><input data-dojo-type="dojox.mobile.TextBox" placeholder="zip code" id="zipCode">
    <br/>
    <br/><input data-dojo-type="dojox.mobile.TextBox" placeholder="Tel no" id="TelNo">
    <br/>
    <br/><input data-dojo-type="dojox.mobile.TextBox" placeholder="E-Mail" id="email">
    <br/>
    <p>Date of birth:</p>
    <input data-dojo-type="dojox.mobile.TextBox" placeholder="DOB" id="dob">
  <br/>
  <button data-dojo-type="dojox.mobile.Button" onClick="processNewCustomer();">Add New Customer</butt
<br/><p>Response:<input data-dojo-type="dojox.mobile.TextBox" placeholder="" id="response" style="width
<br/>
</div>
<!--application UI goes here-->

</div>
<script src="js/initOptions.js"></script>
<script src="js/CICS_Demo.js"></script>
<script src="js/messages.js"></script>

```

Figure 12-11 CICS_Demo.html file

3. The project was created and configured to use Dojo mobile (this example is using Dojo mobile widgets where applicable). The code snippet in Figure 12-11 contains Dojo mobile widgets. Next, alter the CICS_Demo.js file to include the processNewCustomer() function. This function is called when the add New Customer button is pressed. This function gathers input from various fields using jQuery, and calls the addCustomerCall() function that was written earlier.

Select the CICS_App folder, then right-click and select **Run As** → **Build all and deploy**. This will compile the project.

4. After the compilation process has completed, go to the following web page to find the CICS_Demo application in the catalog:

http://localhost:10080/CICS_Demo/console/#catalog

5. To deploy the CreateNewCustomer adapter, right-click the adapters/CreateNewCustomer folder, and select **Run As** → **1 Deploy Worklight Adapter**. See Figure 12-12.

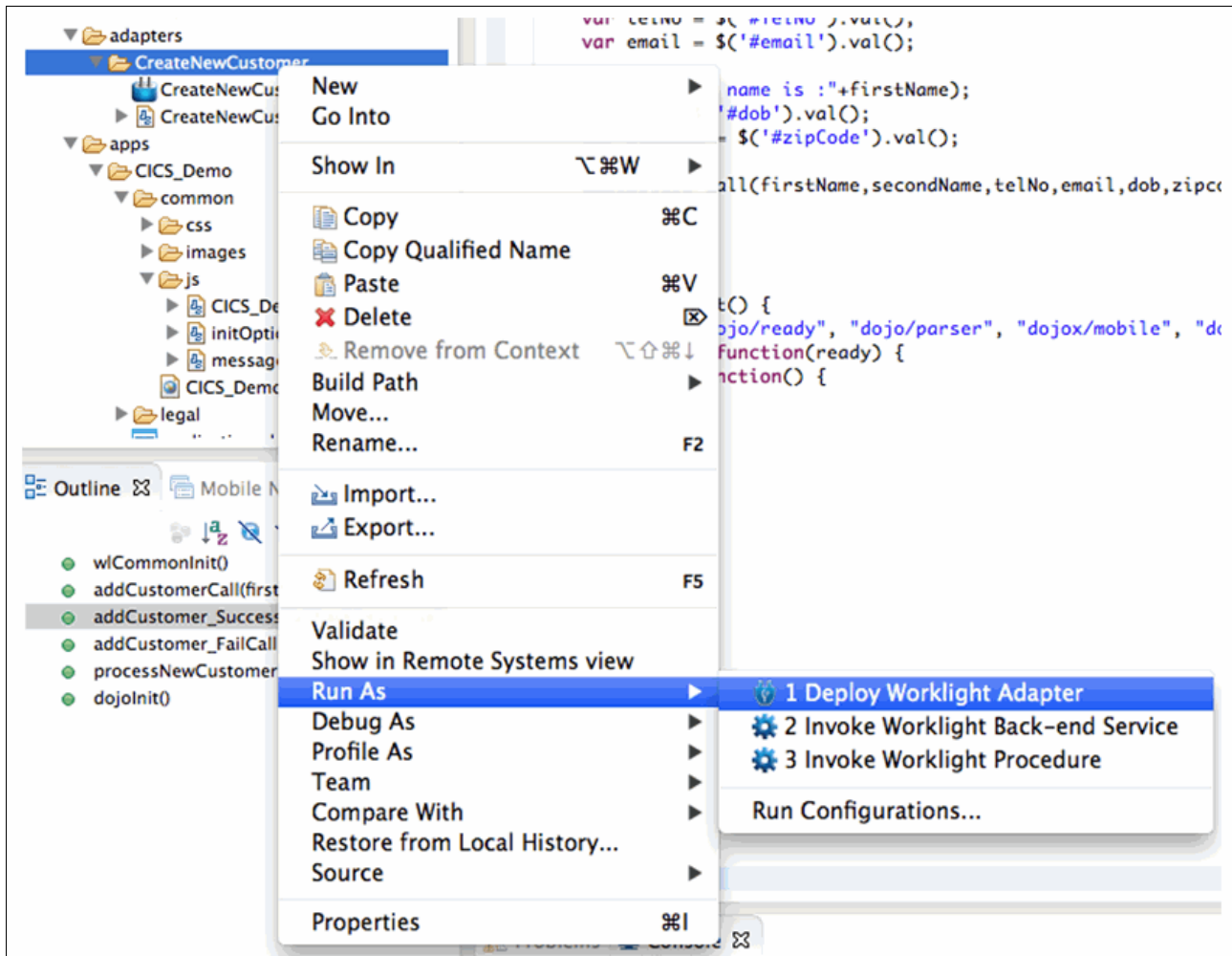


Figure 12-12 Deploy Worklight adapter

6. Refresh the console web page and notice that the console now shows the CreateNewCustomer adapter.
7. Click **Preview** to open a mobile simulator within the web browser, then complete the fields.

8. Click the button to receive the **Response** URL in the text box, as shown in Figure 12-13.



The screenshot shows a web application titled "CICS Demo". It features a vertical stack of input fields and a button. The fields contain the following text: "Andy", "Armstrong", "10101", "07771330510", "andyarmstrong@hotmail", "Date of birth:", "1987-04-12". Below these is a button labeled "Add New Customer". At the bottom, there is a label "Response:" followed by a text box containing the URL "/genapp/CustService/0001000186".

Figure 12-13 CICS Worklight demonstration

This example is designed to provide quick how-to steps to set up a Worklight project to call a back-end service hosted in CICS.



Part 4

Appendix

This part of the book includes Appendixes.



A

Sample level for a JSON schema

This appendix contains the full JavaScript Object Notation (JSON) schema produced by DFHLS2JS as described in Chapter 9, “Language structure to JSON schema scenario” on page 93.

Sample JSON schema generated from COBOL customer create program

Example A-1 shows the full JSON schema that is output from the DFHLS2JS Assistant when run against the general insurance customer create Common Business Oriented Language (COBOL) copybook.

Example A-1 JSON Request schema produced from DFHLS2JS Assistant for Customer Create

```
{
  "$schema": "http://\json-schema.org/draft-04/schema#",
  "description": "Request schema for the LGACUS01 JSON interface",
  "type": "object",
  "properties": {
    "LGACUS01operation": {
      "type": "object",
      "properties": {
        "ca": {
          "type": "object",
          "properties": {
            "ca_request_id": {
              "type": "string",
              "maxLength": 6
            },
            "ca_return_code": {
              "type": "integer",
              "maximum": 99,
              "minimum": 0
            },
            "ca_customer_num": {
              "type": "integer",
              "maximum": 9999999999,
              "minimum": 0
            },
            "ca_first_name": {
              "type": "string",
              "maxLength": 10
            },
            "ca_last_name": {
              "type": "string",
              "maxLength": 20
            },
            "ca_dob": {
              "type": "string",
              "maxLength": 10
            }
          }
        },
        "ca_house_name": {
          "type": "string",
          "maxLength": 20
        },
        "ca_house_num": {
          "type": "string",
          "maxLength": 4
        },
        "ca_postcode": {
```

```

        "type":"string",
        "maxLength":8
    },
    "ca_num_policies":{
        "type":"integer",
        "maximum":999,
        "minimum":0
    },
    "ca_phone_mobile":{
        "type":"string",
        "maxLength":20
    },
    "ca_phone_home":{
        "type":"string",
        "maxLength":20
    },
    "ca_email_address":{
        "type":"string",
        "maxLength":100
    },
    "ca_policy_data":{
        "type":"string",
        "maxLength":30000
    }
},
"required":[
    "ca_request_id",
    "ca_return_code",
    "ca_customer_num",
    "ca_first_name",
    "ca_last_name",
    "ca_dob",
    "ca_house_name",
    "ca_house_num",
    "ca_postcode",
    "ca_num_policies",
    "ca_phone_mobile",
    "ca_phone_home",
    "ca_email_address",
    "ca_policy_data"
]
}
},
"required":[
    "ca"
]
}
},
"required":[
    "LGACUSO10operation"
]
}
}

```



B

Sample COBOL programs

This appendix contains the complete source code for the sample Common Business Oriented Language (COBOL) programs referred to in this book. The source code is also available to download separately. For more information about how to obtain these additional materials, see Appendix C, “Additional material” on page 175.

Sample programs for CICS as a client for JSON web services

The Customer Information Control System (CICS) programs in this section are referred to in Chapter 11, “Developing a simple JSON web service client application” on page 127. They consist of a sample client application that calls a JavaScript Object Notation (JSON) web service, and a service provider application to test the client.

Sample client application

This section contains a sample COBOL program (Example B-1) that demonstrates using the linkable interface to transform JSON and WEB application programming interface (API) commands to call a JSON web service. It calls a sample provider application, which is supplied in “Sample provider application” on page 173.

For more information about how the program works, see 11.5, “Developing the application program” on page 142. For information about how to test the program, see 11.6, “Testing the sample application” on page 148.

Example B-1 Sample client application

```
CBL CICS('COBOL3') APOST
*****
*                                                                 *
*  MODULE NAME = REQUeST                                         *
*                                                                 *
*  DESCRIPTIVE NAME = Sample program demonstrating CICS         *
*                    as a client for a JSON web service         *
*  @BANNER_START@                                           02 *
*                                                                 *
*  Licensed Materials - Property of IBM                         *
*                                                                 *
*  "Restricted Materials of IBM"                                *
*                                                                 *
*  (C) Copyright IBM Corp. 2013                                *
*                                                                 *
*                                                                 *
*  @BANNER_END@                                               *
*                                                                 *
*                                                                 *
*  TRANSACTION NAME = n/a                                       *
*                                                                 *
*  -----                                                    *
*  ENTRY POINT = REQUEST                                         *
*  -----                                                    *
*                                                                 *
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. REQUEST.
```



```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*-----*
* Common defintions                                     *
*-----*
01 COMPLETED-MSG.
03 INITIAL-TEXT PIC X(20) VALUE 'INSURANCE SCORE WAS '.
03 SCORE-TEXT PIC X(3).

* Data structures to hold the input and output data
01 REQUEST-DATA.
COPY SCREQ01.
01 RESPONSE-DATA.
COPY SCRESP01.

01 WORKING-VARIABLES.
03 TRANS-RESP PIC S9(8) COMP.
03 CICS-RESP PIC S9(8) COMP.
03 CICS-RESP2 PIC S9(8) COMP.
03 HTTP-RESP PIC S9(4) COMP.
03 TOKEN PIC S9(16) COMP.
03 ERROR-LENGTH PIC S9(8) COMP.
03 BAD-TRANS-RESP.
05 MSG-TEXT PIC X(48) VALUE
'An error occurred when transforming JSON, code: '.
05 ERROR-DISPLAY PIC X(8).
03 BAD-CICS-RESP PIC X(47) VALUE
'An unexpected error occurred in a CICS command.'.
03 BAD-WEB-RESP PIC X(52) VALUE
'An error occurred connected to the JSON web service.'.
03 BAD-URIMAP PIC X(26) VALUE 'URIMAP could not be found.'.
03 HTTP-MSG.
05 MSG-TEXT PIC X(19) VALUE 'BAD HTTP RESPONSE: '.
05 HTTP-RESP-DISPLAY PIC XXXX.
05 GAP PIC X VALUE IS SPACES.
05 HTTPSTATUS PIC X(50).
03 CONTENT-TYPE PIC X(56) VALUE 'application/json'.
03 ERROR-MSG PIC X(256).

*-----*

*****
* LINKAGE SECTION
*****

LINKAGE SECTION.

*****
* PROCEDURES
*****

PROCEDURE DIVISION.

*-----*

```

MAINLINE SECTION.

```
*-----*
* Common code                                     *
*-----*

      INITIALIZE TRANS-RESP
      INITIALIZE CICS-RESP

      MOVE 'JOE' TO FIRSTNAME
      MOVE 'BLOGGS' TO LASTNAME
      MOVE 67 TO HOUSENUMBER
      MOVE '10/10/1984' TO DOB
      MOVE 'N00 BDY' TO POSTCODE
      MOVE 3 TO POLICYTYPE

      EXEC CICS PUT CONTAINER('DFHJSON-TRANSFRM')
              CHANNEL('CHAN')
              FROM('SCOREREQ')
              CHAR
              RESP(CICS-RESP)
              END-EXEC
      PERFORM CHECK-RESP

      EXEC CICS PUT CONTAINER('DFHJSON-DATA')
              CHANNEL('CHAN')
              FROM(REQUEST-DATA)
              RESP(CICS-RESP)
              END-EXEC
      PERFORM CHECK-RESP

*
      Link to the transformer
      EXEC CICS LINK PROGRAM('DFHJSON')
              CHANNEL('CHAN')
              RESP(CICS-RESP)

      END-EXEC
      PERFORM CHECK-RESP

      PERFORM HANDLE-ERROR

      EXEC CICS WEB OPEN
              URIMAP('CREDSERV')
              SESSTOKEN(TOKEN)
              RESP(CICS-RESP)
              RESP2(CICS-RESP2)
              END-EXEC
      PERFORM CHECK-RESP-WEB

      EXEC CICS WEB CONVERSE
              URIMAP('CREDSERV') POST
              CONTAINER('DFHJSON-JSON')
              CHANNEL('CHAN')
              MEDIATYPE(CONTENT-TYPE)
              TOCONTAINER('DFHJSON-JSON')
              TOCHANNEL('CHAN')
              STATUSCODE(HTTP-RESP)
```

```

        STATUSTEXT(HTTPSTATUS)
        SESSTOKEN(TOKEN)
        RESP(CICS-RESP)
        RESP2(CICS-RESP2)
        END-EXEC
PERFORM CHECK-RESP-WEB

IF HTTP-RESP NOT EQUAL 200
    MOVE HTTP-RESP TO HTTP-RESP-DISPLAY
    EXEC CICS SEND TEXT FROM(HTTP-MSG)
        ERASE END-EXEC
    EXEC CICS RETURN END-EXEC
END-IF

EXEC CICS WEB CLOSE SESSTOKEN(TOKEN) END-EXEC

EXEC CICS DELETE CONTAINER('DFHJSON-DATA')
    CHANNEL('CHAN')
    END-EXEC

EXEC CICS PUT CONTAINER('DFHJSON-TRANSFRM')
    CHANNEL('CHAN')
    FROM('SCORERESP')
    RESP(CICS-RESP)
    CHAR
END-EXEC
PERFORM CHECK-RESP

*
Link to the transformer
EXEC CICS LINK PROGRAM('DFHJSON')
    CHANNEL('CHAN')
    RESP(CICS-RESP)
END-EXEC
PERFORM CHECK-RESP

PERFORM HANDLE-ERROR

EXEC CICS GET CONTAINER('DFHJSON-DATA') CHANNEL('CHAN')
    INTO(RESPONSE-DATA)
    RESP(CICS-RESP)
    END-EXEC.
PERFORM CHECK-RESP

MOVE SCORE TO SCORE-TEXT

EXEC CICS SEND TEXT FROM(COMPLETED-MSG) JUSLAST
    END-EXEC
EXEC CICS SEND PAGE END-EXEC

EXEC CICS RETURN END-EXEC.

EXIT.

HANDLE-ERROR.
EXEC CICS GET CONTAINER('DFHJSON-ERROR') CHANNEL('CHAN')
```

```

        INTO(TRANS-RESP)
        RESP(CICS-RESP)
        END-EXEC
    IF CICS-RESP EQUAL DFHRESP(NORMAL)
*      Error container is present, output value
        MOVE TRANS-RESP TO ERROR-DISPLAY
        EXEC CICS SEND TEXT FROM(BAD-TRANS-RESP)
            ERASE END-EXEC

        MOVE 256 TO ERROR-LENGTH

        EXEC CICS GET CONTAINER('DFHJSON-ERRORMSG')
            CHANNEL('CHAN')
            INTO(ERROR-MSG)
            RESP(CICS-RESP)
            FLENGTH(ERROR-LENGTH)
            END-EXEC

        IF CICS-RESP EQUAL DFHRESP(NORMAL)
            DISPLAY ERROR-MSG
        END-IF
        EXEC CICS RETURN END-EXEC
    END-IF
    EXIT.

CHECK-RESP.
    IF CICS-RESP NOT EQUAL DFHRESP(NORMAL)
        EXEC CICS SEND TEXT FROM(BAD-CICS-RESP)
            ERASE
            END-EXEC
        EXEC CICS RETURN END-EXEC
    END-IF
    EXIT.

CHECK-RESP-WEB.
    IF CICS-RESP NOT EQUAL DFHRESP(NORMAL)
        IF CICS-RESP EQUAL DFHRESP(NOTFND)
            AND CICS-RESP2 EQUAL 1
                EXEC CICS SEND TEXT FROM(BAD-URIMAP)
                    ERASE
                    END-EXEC

            ELSE
                EXEC CICS SEND TEXT FROM(BAD-WEB-RESP)
                    ERASE
                    END-EXEC

            END-IF
        EXEC CICS RETURN END-EXEC
    END-IF
    EXIT.

```

Sample provider application

This section contains a program (Example B-2) that can be used as a JSON web service provider to test the sample client application.

Example B-2 Sample provider application

```
CBL CICS('COBOL3') APOST
*****
*                                                                 *
*  MODULE NAME = CREDIT                                         *
*                                                                 *
*  DESCRIPTIVE NAME = Service provider application for          *
*                    insurance credit score service             *
*  @BANNER_START@                                           02 *
*                    CREDIT                                     *
*  Licensed Materials - Property of IBM                       *
*                                                                 *
*  "Restricted Materials of IBM"                               *
*                                                                 *
*  (C) Copyright IBM Corp. 2013                               *
*                                                                 *
*                                                                 *
*  @BANNER_END@                                               *
*                                                                 *
*  TRANSACTION NAME = n/a                                     *
*                                                                 *
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. CREDIT.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*-----*
* Common defintions *
*-----*

01 CUSTID-SEED PIC 9(9).
01 SCORE-SEED PIC 9(9).

* Data structures to hold the input and output data
* Due to copy books containing 'SYNC' members must be held
* individually with an 01 level structure to ensure they are
* aligned on a double word boundry
01 REQUEST-CONTAINER-DATA.
   COPY CRREQ01.
01 RESPONSE-CONTAINER-DATA.
   COPY CRRESP01.

*-----*
```

```

*****
*   L I N K A G E   S E C T I O N
*****
LINKAGE SECTION.

*****
*   P R O C E D U R E S
*****
PROCEDURE DIVISION.

*-----*
MAINLINE SECTION.

*-----*
* Get the input data from the supplied container      *
*-----*

      EXEC CICS GET CONTAINER('DFHWS-DATA')
              INTO(REQUEST-CONTAINER-DATA)
      END-EXEC

      COMPUTE SCORE-SEED = POLICYTYPE + CUSTID-SEED
      COMPUTE SCORE = FUNCTION RANDOM(SCORE-SEED) * 900 + 100

      COMPUTE CUSTID-SEED = FUNCTION NUMVAL(HOUSENUMBER)
      COMPUTE CUSTOMERID = FUNCTION RANDOM(CUSTID-SEED) * 90000000
      ADD 10000000 TO CUSTOMERID

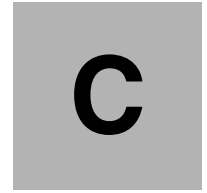
      EXEC CICS ASKTIME ABSTIME(TIMESTAMP) END-EXEC

      EXEC CICS PUT CONTAINER('DFHWS-DATA')
              FROM(RESPONSE-CONTAINER-DATA)
      END-EXEC

* Return to caller
      EXEC CICS RETURN END-EXEC.

MAINLINE-EXIT.
      EXIT.
*-----*

```



Additional material

This appendix refers to additional material that can be downloaded from the Internet, as described in the following sections.

Locating the web material

The web material associated with this book is available in softcopy on the Internet from the IBM Redbooks web server. To download it, go to the following website:

<ftp://www.redbooks.ibm.com/redbooks/SG248161>

Alternatively, you can go to the IBM Redbooks website:

ibm.com/redbooks

Select the **Additional materials**, and open the directory that corresponds with the IBM Redbooks form number, SG24-8161.

Using the web material

The additional web material that accompanies this book includes the following file:

File name	Description
SG248161.zip	Compressed code samples

Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and extract the contents of the web material .zip file into this folder.

Related publications

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

IBM Redbooks publications

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only:

- ▶ *Strategic Overview of WebSphere Appliances*, REDP-4790
- ▶ *Connecting Your Business to the Multichannel Customer with freedomone and IBM Worklight*, REDP-4986
- ▶ *Enabling Mobile Apps with IBM Worklight Application Center*, REDP-5005
- ▶ *Extending Your Business to Mobile Devices with IBM Worklight*, SG24-8117
- ▶ *CICS and SOA: Architecture and Integration Choices*, SG24-5466
- ▶ *CICS Web Services Workload Management and Availability*, SG24-7144
- ▶ *Securing CICS Web Services*, SG24-7658
- ▶ *Securing Your Mobile Business with IBM Worklight*, SG24-8179
- ▶ *Enterprise Caching in a Mobile Environment*, TIPS0953
- ▶ *Getting Started with IBM Worklight*, TIPS1009
- ▶ *Enhancing Your Mobile Enterprise Security with IBM Worklight*, TIPS1054

How to get IBM Redbooks publications

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

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



Implementing IBM CICS JSON Web Services for Mobile Applications

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



Implementing IBM CICS JSON Web Services for Mobile Applications



Includes architectural patterns and example scenarios

Illustrates integration with IBM Worklight

Is based on CICS TS Feature Pack for Mobile Extensions

This IBM Redpaper Redbooks publication provides information about how you can connect mobile devices to IBM Customer Information Control System (CICS) Transaction Server (CICS TS), using existing enterprise services already hosted on CICS, or to develop new services supporting new lines of business. This book describes the steps to develop, configure, and deploy a mobile application that connects either directly to CICS TS, or to CICS via IBM Worklight Server. It also describes the advantages that your organization can realize by using Worklight Server with CICS.

In addition, this Redbooks publication provides a broad understanding of the new CICS architecture that enables you to make new and existing mainframe applications available as web services using JavaScript Object Notation (JSON), and provides support for the transformation between JSON and application data. While doing so, we provide information about each resource definition, and its role when CICS handles or makes a request.

We also describe how to move your CICS applications, and business, into the mobile space, and how to prepare your CICS environment for the following scenarios:

- ▶ Taking an existing CICS application and exposing it as a JSON web service
- ▶ Creating a new CICS application, based on a JSON schema
- ▶ Using CICS as a JSON client

This Redbooks publication provides information about the installation and configuration steps for both Worklight Studio and Worklight Server. Worklight Studio is the Eclipse interface that a developer uses to implement a Worklight native or hybrid mobile application, and can be installed into an Eclipse instance. Worklight Server is where components developed for the server side (written in Worklight Studio), such as adapters and custom server-side authentication logic, run.

CICS applications and their associated data constitute some of the most valuable assets owned by an enterprise. Therefore, the protection of these assets is an essential part of any CICS mobile project. This Redbooks publication, after a review of the main mobile security challenges, outlines the options for securing CICS JSON web services, and reviews how products, such as Worklight and IBM DataPower, can help. It then shows examples of security configurations in CICS and Worklight.

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

**BUILDING TECHNICAL
INFORMATION BASED ON
PRACTICAL EXPERIENCE**

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

**For more information:
ibm.com/redbooks**

SG24-8161-00

ISBN 0738438901