

Extending Your Business to Mobile Devices with IBM Worklight

See how to build, run, manage, and integrate mobile applications with IBM Worklight

Explore how to quickly integrate mobile applications with cloud services

Learn to use IBM Worklight for developing mobile applications



Andreas Dannhauer
Ming Zhe Huang
Paul Idstein
Todd Kaplinger
Hossam Katory
Christian Kirsch
Kearan McPherson
Leonardo Olivera
Susan Hanson

Redbooks



International Technical Support Organization

**Extending Your Business to Mobile Devices with IBM
Worklight**

August 2013

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

First Edition (August 2013)

This edition applies to:

- ▶ IBM Worklight, Version 5.0.5
- ▶ IBM WebSphere® eXtreme Scale, Version 8.5
- ▶ IBM WebSphere Cast Iron Hypervisor Edition Version 6.1
- ▶ IBM WebSphere DataPower XG45 appliance

Contents

Notices	vii
Trademarks	viii
Preface	ix
Authors	x
Now you can become a published author, too!	xi
Comments welcome	xii
Stay connected to IBM Redbooks	xii
Part 1. Business introduction	1
Chapter 1. Understanding the mobile industry	3
1.1 The evolution to mobile technologies	4
1.2 Mobile potential	5
1.2.1 Business-to-consumer	5
1.2.2 Business-to-employee	6
1.2.3 Business-to-business	7
1.3 Mobile challenges	7
1.3.1 Business challenges	7
1.3.2 Concept and design challenges	8
1.3.3 Development challenges	9
1.3.4 Infrastructure challenges	10
1.3.5 Security challenges	11
1.4 Defining a successful mobile strategy	12
1.4.1 Mobile transformation	13
1.4.2 Mobile scenarios and user groups	14
1.4.3 Mobile device selection	15
1.4.4 Mobile applications	16
1.4.5 Mobile integration	17
1.4.6 Mobile operations	18
1.4.7 Finalizing your mobile strategy	20
Chapter 2. Exploring the solution architecture	21
2.1 Understanding the concepts behind IBM MobileFirst	22
2.1.1 Building mobile applications	22
2.1.2 Connecting and running mobile systems	24
2.2 Addressing mobile challenges	25
2.2.1 Business challenges	25
2.2.2 Concept and design challenges	26
2.2.3 Development challenges	27
2.2.4 Infrastructure challenges	29
2.2.5 Security challenges	29
2.3 Understanding the products used in the solution	32
2.3.1 IBM Worklight	32
2.3.2 WebSphere CastIron Hypervisor Edition	40
2.3.3 IBM WebSphere DataPower appliances	41
2.3.4 IBM WebSphere eXtreme Scale and the IBM WebSphere DataPower XC10 caching appliance	42
2.4 Adapting your mobile solution to a different audience	43

2.4.1 Mobile enterprise application platform configuration.	43
2.4.2 Mobile consumer application platform configuration.	45
Chapter 3. Expanding the enterprise using mobile technology	47
3.1 Introducing Airline Company A	48
3.2 IT department roles	48
3.2.1 Previous roles	48
3.2.2 New roles to support mobile technologies	49
3.3 Business goals and performance indicators	50
3.4 Building the mobile roadmap.	51
3.4.1 Mobile transformation (building the roadmap).	51
3.4.2 Mobile scenarios and user groups (who will use the app and how)	52
3.5 Building the mobile strategy	53
3.5.1 Mobile device selection.	53
3.5.2 Mobile applications	53
3.5.3 Mobile integration	53
3.5.4 Mobile operations	54
3.6 Selecting a Mobile Application Platform	54
Part 2. Scenario introduction and technical implementation	57
Chapter 4. Creating and deploying the mobile solution	59
4.1 Breaking down the scenario	61
4.1.1 High-level use case.	61
4.1.2 Agile breakdown and design.	62
4.1.3 Components of mobile luggage tracking solution	65
4.1.4 Affected roles in Airline Company A's IT organization	67
4.2 Planning and setting up the environments	67
4.2.1 Setting up the development environment	68
4.2.2 Setting up the pre-production environment.	69
4.3 Providing enterprise services	71
4.3.1 Services definition.	71
4.3.2 XML Data Format	73
4.3.3 Services implementation.	74
4.4 Planning for security	75
4.4.1 Understanding the mobile authentication flow	75
4.4.2 Planning client-side authentication	76
4.4.3 Planning server-side authentication	77
4.4.4 Pulling it all together	78
4.5 Operating the Mobile Enterprise Application Platform	78
4.5.1 Managing access to the Application Center	78
4.5.2 Managing access to mobile applications.	81
4.5.3 Deploying the Worklight Application Center mobile client.	84
4.6 Introducing Worklight Studio projects	87
4.7 Creating the adapters	88
4.7.1 Setting up the adapter project.	88
4.7.2 Creating the authentication adapter	91
4.7.3 Creating the business services adapter	99
4.8 Creating the mobile application.	104
4.8.1 Creating the user interface	105
4.8.2 Creating the shell for the bar code scanner	121
4.8.3 Creating the client-side authentication component	140
4.8.4 Integrating the solution	142
4.8.5 Unit testing the application	150

4.8.6 Building the mobile application	160
4.9 Deploying the application	170
4.9.1 Deploying the hybrid application packages.	170
4.9.2 Deploying the project-specific web application archive.	174
4.9.3 Deploying the Worklight application	176
4.9.4 Deploying the Worklight adapters	178
4.10 Testing the end-to-end scenario	179
4.10.1 Installing the Application Center mobile client.	180
4.10.2 Installing the mobile application	185
4.10.3 Starting the application and performing the tests	188
4.11 Using feedback mechanisms	189
4.12 Analyzing application and adapter usage	191
4.12.1 Configuring the reporting feature	192
4.12.2 Setting up the reporting environment	193
4.12.3 Running the reports	199
Chapter 5. Expanding the solution to consumers	205
5.1 Breaking down the new scenario	207
5.1.1 High-level use case.	207
5.1.2 Agile breakdown and design.	207
5.1.3 Solution components	211
5.1.4 Expanding the topology	212
5.1.5 New tasks to implement the customer solution.	214
5.2 Changes to the environments	215
5.2.1 Updating the development environment	215
5.2.2 Changes to the pre-production environment.	220
5.3 Changes to the back-end services	232
5.3.1 Creating the hotel adapter	232
5.3.2 Updating the authentication adapter	237
5.4 Creating the new mobile application	239
5.4.1 Creating the user interface using Dojo	240
5.4.2 Developing the native map page	264
5.4.3 Integrating the map into the mobile application.	274
5.4.4 Adding the new authentication mechanism.	281
5.4.5 Building the new mobile application	284
5.5 Unit testing	290
5.6 Deploying the application	291
Chapter 6. Installation and configuration	293
6.1 Installing products with IBM Installation Manager	294
6.2 Installing IBM Worklight Server	295
6.2.1 Performing the installation	296
6.2.2 Verifying the installation by starting Worklight Server.	298
6.2.3 Verifying the installation with IBM Worklight Console	299
6.2.4 Verifying the installation with IBM Application Center.	300
6.3 Installing IBM Worklight Studio	300
6.3.1 Installing IBM Worklight Studio on Mac OS X.	301
6.3.2 Installing the Android SDK on Mac OS X	303
6.3.3 Installing IBM Worklight Studio on Windows XP	308
6.4 Installing WebSphere Application Server Liberty Profile	309
6.4.1 Installing the Liberty Profile.	309
6.4.2 Creating the application server instance.	311
6.5 Installing IBM WebSphere Cast Iron	315

6.5.1 Deploying the server component (virtual appliance template)	316
6.5.2 Installing the developer studio	316
6.6 Installing IBM WebSphere eXtreme Scale	319
Appendix A. Mobile solution additional source	321
Worklight Server server.xml file	323
Back-end service source	325
Java package com.ibm.itso.saw210.beans.	325
com.ibm.itso.saw210.dao	329
com.ibm.itso.saw210.facade.	330
com.ibm.itso.saw210.services.rest	331
com.ibm.itso.saw210.servlet.	333
com.ibm.itso.saw210.util.	334
Deployment descriptor	334
LuggageTracker application source files	335
Business services adapter files.	335
Authentication adapter files.	337
User registry	339
LuggageTracker mobile application files.	341
Bar code scanning files.	348
MyLuggage application source files	350
Business services adapter files.	350
Authentication adapter files.	353
Map adapter files	354
WebSphere eXtreme Scale caching integration files	356
Dojo mobile application source files	360
Native map files	373
Appendix B. Additional material	379
Locating the web material	379
Using the web material.	379
Downloading and extracting the web material	379
Related publications	381
IBM Redbooks	381
Online resources	381
Help from IBM	383

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not 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 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:

Cast Iron®
Cognos®
DataPower®
developerWorks®

IBM®
Lotus®
Rational®
Redbooks®

Redbooks (logo) ®
Tealeaf®
Tivoli®
WebSphere®

The following terms are trademarks of other companies:

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

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

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

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

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

Preface

The mobile industry is evolving rapidly. An increasing number of mobile devices, such as smartphones and tablets, are sold every year and more people are accessing services from a mobile device than ever before. For an enterprise, this can mean that a growing number of customers, business partners, and even employees now expect to access services on a mobile channel. This opens new opportunities for the business but also presents new challenges, both in terms of business processes and information technology (IT) infrastructure.

IBM® Worklight® is an open and extensible mobile application platform that brings together many mobile capabilities into a single product. It helps organizations of all sizes to efficiently develop, connect, run, and manage HTML5, hybrid, and native applications. It uses standards-based technologies and tools, supports multiple security mechanisms, and delivers a comprehensive development environment, mobile-optimized middleware, and an integrated management and analytics console. IBM Worklight provides the essential elements needed for complete mobile application development, deployment, and management within a business.

IBM Worklight enables the creation of rich, cross-platform applications without the need for code translation or proprietary interpreters. It helps an IT organization reduce the time to market, cost, and complexity of development, and enables better user experiences across the range of mobile devices.

This IBM Redbooks® publication provides information necessary to design, develop, deploy, and maintain mobile applications using IBM Worklight, IBM WebSphere® Cast Iron®, and other key IBM products. It includes information about decision points that the IT organization will need to make, the roles that are involved in a mobile strategy and the responsibilities of the individuals in those roles. It also describes integration points with other IBM products that can enhance the mobile solution.

This book is separated into two parts:

- Part 1 is intended for a more business-oriented information technology (IT) audience and addresses the business aspects of the mobile industry. The chapters in Part 1 focus on the considerations and tasks that are necessary to build a mobile strategy and explains how decisions, such as which mobile devices to support can affect various organizations within an enterprise. The target audience for this part of the book is an IT architect or CTO, who can translate business needs into information technology solutions. These individuals have knowledge of both business and technical aspects of the enterprise and collaborate with other staff and stakeholders about establishing business goals. They then help translate these goals into the requirements for new and existing IT projects, systems, and infrastructure.
- Part 2 is intended for a technical audience, including application developers, testers, and system administrators. It details a step-by-step process to reuse existing business services and extend those services to a mobile device. The chapters in this part discuss the actual coding, building, and testing of a mobile application using IBM Worklight Studio, including testing using both simulated and real devices. Deployment of the application to an application store where users can install the mobile application, and updating the application to fix problems and add new functionality will also be covered.

Authors

This book was produced by a team of specialists from around the world working at the IBM China Development Lab, Shanghai, People's Republic of China.

Andreas Dannhauer is an Architect in the IBM Software Group's Lab Services in Boeblingen, Germany. He is currently on assignment in the Shanghai branch of the China Development Lab to drive the mobile business in the greater China region. He earned his degree in Business Informatics, and specialized in mobile, wireless, and embedded computing after joining IBM in 2001. He has deep insight into the automotive and retail industries and specializes in architecting and implementing end-to-end solutions for mobile and distributed environments. He was a key driver in various mobile, telematics and retail projects based on IBM WebSphere and Lotus® embedded software technology.

Ming Zhe Huang is a member of the IBM Worklight development team from the IBM China Software Development Lab, Shanghai, People's Republic of China. He has five years experience in Web 2.0 development including AJAX and Flex. Prior to joining IBM, he worked on various Web 2.0 projects including SaaS, visualization, and mobile web development. He works closely with various customers to architect, design, and build mobile solutions for their businesses.

Paul Idstein is an IT Service Specialist with IBM Software Group WebSphere Services in Germany. He has four years of experience in mobile computing and web technologies working for variety of customers in automotive, manufacturing, insurance, and banking industries. He holds a degree in International Business Information Technology from Baden-Wuerttemberg Corporate State University. His areas of expertise include distributed web systems, mobile messaging, and connectivity. He has written extensively about business intelligence and cloud computing.

Todd Kaplinger is a Senior Technical Staff Member (STSM) and IBM Master Inventor in the IBM Software Group. He is the Mobile Cloud Platform Architect (Worklight - IBM MobileFirst) and has been a leading IBM Thought Leader and Architect in mobile for the past three years. Todd is an expert in web-based technologies such as Dojo, JSP, Servlet, and PHP, with recent focus on emerging Web 2.0 technologies and their impact on the enterprise. Todd has been the Lead Architect on other WebSphere projects and has participated in the JSR 154 Servlet 2.5 Specification as an IBM representative in the Servlet Expert Group.

Hossam Katory is an IT Specialist with the IBM Software Group Cairo Lab in Egypt. He has eight years of experience in software globalization and the bidirectional languages support field. He holds a Bachelor of Engineering degree in Computer Engineering from the Arab Academy for Science, Technology, & Maritime Transport. His areas of expertise include EAM and ISM solutions, and also mobile web technologies. He has written extensively about building mobile web applications.

Christian Kirsch is a certified Senior IT Specialist with the IBM Software Group, working in the Mobile Business Solution Services Team in Boeblingen, Germany. In the last 13 years, he has gained experiences by leading and developing mobile solutions for customers in different brands and industries. His areas of expertise include mobile application development, workforce mobility, remote access, asset monitoring, track and trace, device management, and mobile remote patient monitoring solutions. Besides his customer engagements, Christian is an author of many articles for magazines, books, and blogs, and also patents about mobile technology.

Kearan McPherson is an IT Specialist in the IBM Software Solutions Lab based in Johannesburg, South Africa. He has four years of experience in virtualization and cloud computing technologies. He holds a Bachelor of Science (Honors) degree in Computer Science from the University of the Western Cape. His areas of expertise include virtualization and service management solutions. He has written extensively about cloud computing

Leonardo Olivera is an IT Specialist with IBM Global Services, Uruguay. He has 17 years of experience in application development and systems integration. He holds a degree in Computer Science Engineering from Universidad Católica del Uruguay. His areas of expertise include Java Enterprise Edition architecture, WebSphere Portal software, and sensor and actuators solutions. The last few years he has been working on mobile solutions for the healthcare industry.

Susan Hanson is a member of the WebSphere Application Server foundation development team. She has 22 years of experience in developing and delivering IBM software products across the WebSphere and IBM Tivoli® brands. She holds a Bachelor's degree in Computer Science from East Carolina University and a Master's degree in Computer Information Systems from The University of Phoenix. She is based in Research Triangle Park, North Carolina and is temporarily working at the IBM China Development Laboratory in Shanghai, People's Republic of China.

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

Tamikia Barrow, Deana Coble, Linda Robinson, Carla Sadtler,
Margaret Ticknor, and Shawn Tooley
International Technical Support Organization, Raleigh Center

Robyn Gold, Frank Haynes
IBM US

Thomas Hesmer, Frank Müller
IBM Boeblingen, Germany

Yu Cao, Xiao Xia Qiu, and Yuan Peng Zhang
IBM China Software Development Lab, Shanghai

Now you can become a published author, too!

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- Send your comments in an email to:

redbooks@us.ibm.com

- Mail your comments to:

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

Stay connected to IBM Redbooks

- Find us on Facebook:

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

- Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- Look for us on LinkedIn:

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

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

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

- Stay current on recent Redbooks publications with RSS Feeds:

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



Part 1

Business introduction

This part introduces the business concepts regarding the mobile industry and is intended for a more business-oriented information technology (IT) audience. High-level concepts behind mobile technology are presented, including the potential of investing in mobile technology and also the challenges that many companies face when first attempting to enter this new business area. It explains how an IT architect or CTO can address these challenges and can help translate the business goals into the requirements on new and existing IT projects, systems, and infrastructure.

This part contains the following chapters:

- ▶ Chapter 1, “Understanding the mobile industry” on page 3
- ▶ Chapter 2, “Exploring the solution architecture” on page 21
- ▶ Chapter 3, “Expanding the enterprise using mobile technology” on page 47



Understanding the mobile industry

This chapter introduces the many opportunities that mobile solutions can bring to your business and also the challenges that come with them. It outlines approaches to overcome these challenges and create a foundation for a successful mobile strategy.

The following topics are covered:

- ▶ 1.1, “The evolution to mobile technologies” on page 4
- ▶ 1.2, “Mobile potential” on page 5
- ▶ 1.3, “Mobile challenges” on page 7
- ▶ 1.4, “Defining a successful mobile strategy” on page 12

1.1 The evolution to mobile technologies

The history of technology is marked by constant evolution and reinvention. The pattern of reinvention is one of extremes, typically seen in cycles of roughly 10 years. As technology advances and new boundaries are established, usage patterns expand and new business models emerge. This section details the evolution to mobile technologies, from the advent of enterprise computing (business-to-employee and business-to-business) to the emergence of today's consumer-driven applications (business-to-consumer).

During the 1980s, most large companies were investing in technology and large scale enterprise computing. In this model, there was a centralized system, normally using a mainframe computer, that was accessed by employees stationed at terminal servers. IT administrators developed and managed applications on a single mainframe platform, and the applications were designed strictly for enterprise employees to use for corporate tasks. So a consumer seeking to purchase an airline ticket had to contact the airline directly, or use a travel agent intermediary with access to the airline's reservation system. Because the cost of terminal servers was not insignificant, the options for finding an agent with access to a particular airline's system were limited.

In the 1990s, the industry shifted toward a client/server model as the power of desktop computers increased and their cost went down. This shift resulted in a significant change in employee workloads and created a model where multi-tasking using multiple applications simultaneously became standard procedure. In the client/server model, the concept of an *application* spanned both the desktop and the server. This added new challenges for the IT department, because it was now responsible for installing and maintaining the application in the enterprise data center, and also for ensuring that it was properly installed and working on each employee's desktop hardware. As with the mainframe model, the client/server model was still heavily weighted toward business to employee scenarios. However, with the advent of early Internet service providers such as AOL, Prodigy, and CompuServe, the industry started to expand its client application delivery model to non-employee consumers. Yet the reach of these applications was still limited and required specialized software to be installed on the users computer. In many cases, these early Internet service providers did not provide full access to the World Wide Web through a browser. To extend the airline ticket scenario, consumers could, in small numbers, book their own airline tickets using the services provided by these early Internet service providers in a manner similar to what a travel agent could do at a terminal.

The 2000s saw a shift back to a centralized model where applications were hosted on enterprise servers but were now accessible from web browsers. This shift was accelerated by the explosion of web-based technologies. Rich Internet applications delivered through next-generation web browsers provided significant productivity improvements among corporate employees. In addition, the cost of administering and maintaining enterprise applications was reduced, because the vast majority of application code could be managed at the enterprise server tier. However, the shift back to a centralized model also resulted in a significant increase in demand for corporate resources such as enterprise servers, network connectivity, and desktop computing power.

In parallel, consumer access to the Internet become pervasive. This resulted in a strong demand shift in which consumers sought increased access to Internet-based services and the concept of *e-commerce* emerged as a new business model. Many corporations, including airlines, quickly reacted to this new demand by offering services over what became known as the *web channel*. There was an immediate rise in next-generation web storefronts such as Expedia and Travelocity, where the ticket reservation process could be completed without any verbal communication between buyer and seller. Previously, however, the number of users that could access enterprise services was limited; now the scope was unbounded in

developed economies and was at least expanding in emerging economies. Consumers could easily book airline tickets from the comfort of their own homes, without having to install specialized software on their home computers. As a result, the use of travel agents for booking trips dropped significantly. The power of the Internet was channeled to the consumer.

Starting in the late 2000s, another technology shift emerged with the introduction of mobile devices. Although most technology trends are driven by the industry itself, the mobile movement is primarily driven by consumer and marketing. Popular applications such as Dropbox, Evernote, and WhatsApp have been boosted by strong consumer interest and were introduced to solve problems that did not, at that time, have a solution for. For example, Dropbox provides a way to share documents between multiple devices, removing the need to use email, thumb drives, or external storage mechanisms while providing a way to have these resources backed up. Other applications have grown through the use of new marketing techniques that are unique to mobile, including the use of application stores and user ratings. User ratings, including general feedback on social networking sites such as Facebook, are especially important in competitive marketplaces where consumers have many choices.

From a technology perspective, the rise of mobile applications has resulted in yet another shift back to the client/server model, where the mobile device becomes the client that is accessing services that are hosted in the enterprise server tier. The significant difference is the distribution model, thanks to the advent of the *application store*, first popularized by Apple's App Store and later implemented by most competing companies (Google Play, BlackBerry World, Windows Phone Marketplace, and so on). The application store concept is focused on engaging the user. Users are now able to download applications from the application store and provide ratings and feedback, voicing their opinion of the application and its value.

There have even been dramatic innovations in mobile device infrastructure and technology in the years since the mobile revolution began. The removal of bottlenecks in terms of device capabilities and network speeds has significantly improved the mobile experience for users. These bottlenecks were previously seen as inhibitors to expanded adoption of mobile technologies, because the Internet requires significant infrastructure to provide high-speed access to the masses in all economies worldwide. The introduction of mobile technologies relaxes this requirement because Internet access for mobile devices only requires centralized cell towers or satellites and there is no need to physically connect each device to a service provider. As a result, the reach of mobile devices and their access to enterprise services has expanded and more mobile devices than desktop devices are connected to the Internet. With mobile applications, consumers can purchase an airline ticket themselves, and they can plan their entire trip, reserve a hotel room, and find a nearby restaurant.

1.2 Mobile potential

Multiple scenarios can take advantage of mobile technology. They are categorized based on the relationship between the enterprise and the targeted user group:

- ▶ Business-to-consumer: The relationship between an enterprise and its customers
- ▶ Business-to-business: The relationship between two or more companies
- ▶ Business-to-employee: The relationship between a company and its employees

1.2.1 Business-to-consumer

Mobile *business-to-consumer* scenarios are mostly aimed at strengthening customer loyalty to the company to create additional revenue. Customers can interact with the company at any

time to retrieve information or access services. The company, in turn, learns new information that it can use to customize additional offerings to meet customer needs and interests.

In addition to this traditional e-commerce-related scenario, mobile technology has much broader potential because it can enhance the customer experience by utilizing context-aware information. Typically, customers in a store are unknown until they are identified at the cash register by, for example, showing a loyalty card. With mobile technology, customers can be recognized as soon as they enter the store or when they retrieve information about a product using the store's mobile application. When recognized, these customers can be informed about current discounts and be directed to the location of their favorite products within the store.

Another way to enhance the customer experience is to make buying goods easier. With traditional magazine advertisements, customers must write or remember the product and the company in order to later purchase the item online or in a store. By adding a bar code to the advertisement, customers can scan the code using a smartphone and instantly purchase the item, pay for it, and arrange to have it shipped to their home. Inside a store, a bar code can help the customer retrieve additional information about the product, such as details about its ingredients or seeing its label translated into a different language.

Adding social aspects to a mobile application, such as the ability to share information on a social network, puts mobile technology in another dimension. In addition to interacting with the customer, a company can now interact with the customer's social network, which can include possible new customers. And when a customer uses the company's mobile application to share something, such as an opinion of a product, the company can easily analyze what was said and quickly respond to any feedback.

1.2.2 Business-to-employee

Mobile *business-to-employee* scenarios facilitate access to business processes and collaboration among employees. Employees get faster, easier access to information so they can make appropriate, informed decisions more quickly. In their simplest form, business-to-employee applications allow mobile communication using email, instant messaging, or enterprise social media. At the next level, mobile-enabled employees are granted access to business information and processes. For example, a sales representative with access to current delivery times for a certain product is able to immediately provide feedback to a customer who has shipping questions.

Business processes can be accelerated with the use of mobile technology. For example, when a particular business need arises, the request can be immediately approved by the responsible party using a mobile channel. Invoices can be triggered as soon as an employee finishes work at the customer location, or as soon as a product is handed over. In addition, the use of mobile technology improves information and data quality because everything is collected and passed on directly, without a break in the process, such as occurs when information on paper must be entered into an electronic system.

In addition to the productivity benefits for the corporation, the ability to use mobile technology as part of a person's job is becoming increasingly important in attracting potential employees. This includes allowing employees to communicate outside of the office and to work using a mobile device when necessary. The ability to access business applications, including email, messaging, and applications specific to a person's job responsibilities, allows them to be more productive and with greater flexibility. Another example is the ability for a repair person to be able to access the parts catalog to determine if a needed part is in-stock while at a customer's home, repairing an appliance such as a dishwasher.

Because of the rapid evolution of mobile technology, it is expensive for companies to equip and upgrade every employee with appropriate mobile devices from the start. Therefore, many companies are examining the possibility of using the mobile devices owned by their employees. This is often referred to as *bring your own device (BYOD)*. Depending on the classification of the information the employee accesses, and the device's abilities to protect this information, BYOD can reduce company costs.

1.2.3 Business-to-business

Mobile *business-to-business* scenarios get little attention today, but this is expected to change. Applications in this category help to improve cooperation between business partners and drive additional business potential. Similar to business-to-employee applications, the goal here is to grant mobile information access to the involved business partners so that they are more efficient and informed, and are able to make appropriate decisions more quickly.

However, creating effective business-to-business applications requires that all parties agree on the mobile technology standards to be used. Successful solutions also require that the involved partners have defined their own mobile strategies before discussing inter-company solutions. Today, companies are mainly focused on developing their own mobile strategy in order to define the foundations for employee-facing or consumer-facing solutions. As soon as these foundations are built, business-to-business applications will begin playing a more important role.

1.3 Mobile challenges

To fully exploit the potential of mobile technology, a variety of challenges must be overcome, including reviewing and reconsidering existing business processes and models. Issues related to security and the establishment of security policies, especially in the business-to-employee and business-to-business environments, must also be addressed. Enterprises also face new challenges in application design and development that are driven by the special characteristics of mobile applications.

1.3.1 Business challenges

A company can use mobile applications to reduce its effective distance from its customers. Most people who own a smartphone carry it with them all the time, unlike a notebook computer, so a company's mobile application is almost always accessible to customers. Customers, in turn, expect the company to respond quickly when they access services and information through the mobile channel. Business processes and staffing levels, therefore, must be designed and planned for this customer expectation.

With the increased use of social media, a company no longer has control over what its customers, business partners, or employees are saying about it and its products. This can present a real threat in terms of brand image, so companies must actively monitor these outlets and respond as needed. Examples include monitoring comments and reviews on the company website, and actively addressing any negative comments that a customer did not tell the company directly but stated in a social media space. A company can also use social media to thank customers for their business. In some cases, a business might not actually know who all of its customers are, especially for a company that sells a product without registering. By using this new channel, the company can connect to these types of unregistered customers.

Mobile-related business challenges might not be limited to only a few business processes. They can touch upon the entire business. In the past, a retailer's logistics were focused on delivering a large number of products to a small number of store locations. Now that same retailer might be delivering only a few products to many locations, including directly to customers. As more products are sold through mobile applications, the importance of a retailer's in-store equipment can lessen while the design and usability of its mobile applications grows more critical.

Therefore, evaluating the capabilities of new mobile technologies and how they might affect your current business model are important. You will need answers to these key questions:

- ▶ How can we benefit from the development of mobile technologies?
- ▶ Which new business models are possible?
- ▶ How will our business gain a competitive advantage?

1.3.2 Concept and design challenges

Mobile devices have different characteristics that must be considered when you develop applications. One of the most obvious is that, unlike the PC, a mobile device has a small screen. Originally this challenge was addressed by reduction: Both the functions of mobile applications and the information they displayed were reduced to the minimum the user required while on the go.

Meanwhile, user behaviors have changed. The term mobile no longer means simply *on the go* but rather *always with me*. For an employee using an internal company application, this might mean in an internal company meeting, visiting a customer, or sitting at home on the couch. For a customer of the enterprise using a customer-facing application, this means that they have access to the application whenever they need it or want it.

This brings new challenges for developers of internal company mobile applications, which now must be easy to use and also give access to many of the same functions and information available through the company's other channels. During the concept and design phases, another important consideration is what functions should be made available through a mobile application to provide the most flexibility and productivity to employees.

The small size of mobile devices is accompanied by limited input capability, so mobile applications should seek to help users input information. Examples include providing easy access to the address book or GPS to help the user enter an address, or displaying a keyboard with the correct characters based on the type of information the user is expected to type. Voice recognition is also growing in importance as an input channel.

1.3.3 Development challenges

Several approaches for developing mobile applications exist. Figure 1-1 summarizes the capabilities of the four most important approaches:

- ▶ Web development
- ▶ Hybrid development
- ▶ Hybrid mixed development
- ▶ Native development

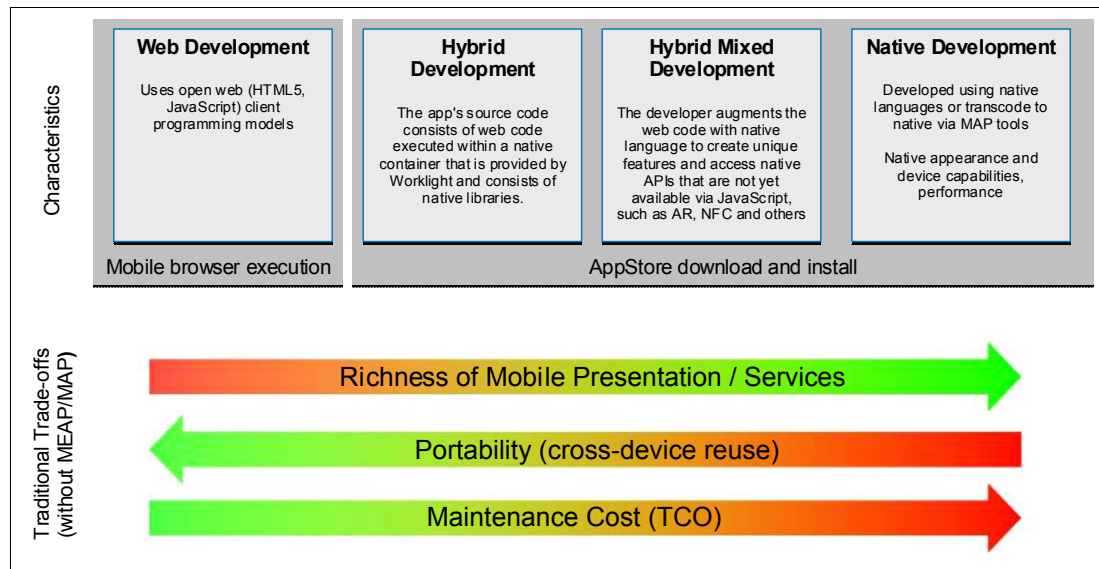


Figure 1-1 Comparison of characteristics of various mobile development approaches

Using *web development*, the application runs inside the browser of the mobile device, using standardized technologies such as HTML5, CSS3 and JavaScript. These application are platform-independent, so supporting a new mobile platform does not always require developing a completely new application. It might only involve small modifications to support a different browser engine. Mobile web applications are not able to access platform functions as they rely purely on the browser and associated web standards. Mobile web applications are not distributed through an application store but are instead accessed through a link on a website or a bookmark in the user's mobile browser.

Applications created using the *hybrid development* approach use parts of both the native development and web development approaches. The hybrid application runs inside a native container and uses the browser engine to display the application interface, which is based on HTML and JavaScript. The container enables access to device capabilities that are not accessible in web applications, such as a smartphone's accelerometer, camera, and local storage. Similar to native applications, hybrid applications are distributed through the platform's application store.

Hybrid mixed development enhances the hybrid development approach. It creates applications that use a container to access device capabilities but also use other native, platform-specific components such as libraries or specific UI elements to enhance the mobile application.

Using *native development*, the application is written for a particular platform and runs on that platform only. The application can make full use of all platform functions such as accessing the camera or contact list or interacting with other applications on the device. Some of the most popular platforms today are Google Android, Apple iOS, BlackBerry OS, and Microsoft

Windows Phone. Supporting each of these platforms requires developing separate applications, possibly using distinct programming languages. Native applications are typically distributed through an application store.

Each of these development approaches has advantages and disadvantages. The specific requirements for an individual mobile solution should drive the selection of an appropriate development approach.

In addition to the design and development of mobile applications, other topics are important also. Depending on the architecture of the application, running certain parts of the business logic as back-end services might be possible. These might have to be developed from the beginning (“from scratch”), but existing services typically can be used or adapted. These services must be integrated into the mobile solution, and developing all of these components to create the solution involves many different skills, not just mobile application development.

Another challenge in developing mobile applications is testing. If the application is intended for multiple platforms, it must be tested on all of them. And for each platform, you might need to test different devices with different functionalities and display sizes. In addition, to run end-to-end tests, the devices need access to a test environment. This poses a problem, because in traditional web development, a test environment is usually protected behind firewalls and not connected to the Internet.

1.3.4 Infrastructure challenges

Just as with mobile application development, the development of mobile application infrastructure presents similar, although slightly different, challenges in terms of availability, scalability, extensibility, reliability, manageability, and performance. Today’s mobile application adoption rates can rise explosively. Consider how fast news of a new mobile application can spread through social media channels. This implies huge business opportunities, but the mobile infrastructure must be able to handle the increase in customer requests.

Mobile users expect instant responses regardless of whether the application is running locally on the device or on a server. Yet mobile networks are usually not as fast or reliable as wired networks. And mobile devices have limited computing power compared to a PC, so processing a request or response adds yet more time before the user can see the result. So the infrastructure must be built to keep the amount of processing performed on the device to the lowest level possible.

In addition, the back-end services that power the mobile application need to be flexible in adapting to the amount of information that can be displayed on mobile screens. Instead of one big request, a more suitable approach might be to have multiple requests that each convey smaller amounts of information.

Some mobile applications might include sending notifications to devices based on user subscriptions for consumer applications, or sending to an employee an alert about a problem with an internal server. These notifications, in many cases, can use services provided by the device vendor and the back-end services will need to access these Internet services. This can require infrastructure changes and must be considered early in the process.

1.3.5 Security challenges

Creating and deploying a new mobile solution involves many groups both inside and outside of the enterprise, and might affect multiple components. To successfully establish end-to-end security, the risks being exposed by each of them must be determined and addressed.

Figure 1-2 depicts the relationship among the groups and components typically involved in a mobile solution.

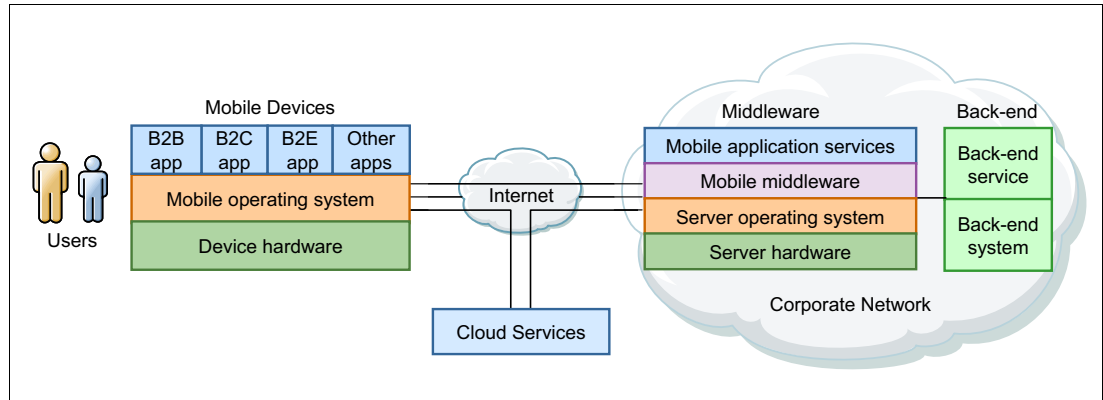


Figure 1-2 Mobile solution overview

Different security challenges are involved in each part of a mobile solution.

Mobile device users

A mobile device is usually used by only one person. However, sharing a device with friends or family members is not uncommon, and these other users might download or install malicious software unintentionally.

In addition, in some business-to-employee or business-to-business scenarios, a mobile device might be shared between colleagues who are currently on duty. So, if certain applications and information cannot be shared between colleagues, there might be a need to distinguish one user from another on the same device.

Mobile devices

A mobile device consists of three critical components:

- ▶ Device hardware
- ▶ Mobile operating system
- ▶ Applications

In terms of hardware, mobile devices, and particularly their memory cards, are small and therefore easily lost, and the person who finds the device or memory card has easy access to the information stored on it. Even when encryption is used to protect the data, there might be special requirements for encryption that cannot be met by a particular device.

Mobile operating systems, similar to desktop operating systems, can contain security holes. Updates can be issued to fix these holes, but depending on the communications provider and the mobile platform provider, a significant amount of time can be spent to distribute these fixes to all users and devices. In addition, some older phones no longer get any updates.

From an application perspective, if internal company information will be stored on a mobile device, define whether or not other applications will be allowed to access this information.

There is a danger that questionable applications might be able to access and distribute sensitive information in a way that is beyond your company's control.

Networks

Similar to traditional network security, you must ensure that information that is transmitted to mobile devices cannot be read by unauthorized individuals or applications. Prior to mobile applications, some employees were able to access the corporate network using a public WiFi hot spot with personal computer or notebook, normally through a VPN. Using a VPN that encrypts information reduces the risk of unauthorized access, but the public WiFi hot spot can introduce risks for malware and viruses.

Mobile devices can also connect to the public WiFi hot spots, but also can use the cellular carrier network for those users with data plans. In addition, the mobile device is normally with you *all of the time* and so users are more likely to connect to the internal network, and more often.

As soon as mobile devices are connected to the internal network in business-to-business a scenario, such as by using VPN or WLAN, the devices become part of the corporate network and the same rules apply to them as apply to the company's desktop PCs and notebook computers.

Mobile middleware

Like traditional middleware, mobile middleware needs to access services outside of the corporate network, such as external cloud services or the notification services of the mobile platform provider. This adds more security challenges, because you must be sure these services are connected securely. Also crucial is to protect the mobile middleware against malicious responses, such as from a compromised cloud service.

Internal enterprise back-end services

Usually, information stored in enterprise back-end systems is classified according to its level of confidentiality. With mobile access, this classification must be extended. It must clearly define which mobile devices are allowed to access various types of classified information based on device ownership (company or personal) and device platforms (Android, BlackBerry, iOS, Windows Phone, and so on, some of which you might consider more secure than others).

External cloud services

Security guidelines also need to establish how to interact with external cloud services, not only from the middleware but also from the mobile devices itself. The seamless integration of mobile devices with cloud services allows easy storage of internal information outside of the corporate network, even by accident.

1.4 Defining a successful mobile strategy

As described in 1.3, "Mobile challenges" on page 7, enterprises face a wide range of challenges when entering the mobile space. Developing a mobile application is only one of them. The following sections describe a domain model or conceptual view that decomposes the mobile space into various domains and can help an enterprise define a successful mobile strategy.

Figure 1-3 shows the domain model for defining a mobile strategy. It is divided into six aspects or domains:

- ▶ Mobile transformation
- ▶ Mobile scenarios and user groups
- ▶ Mobile device selection
- ▶ Mobile applications
- ▶ Mobile integration
- ▶ Mobile operations

Each domain addresses a specific topic and presents certain questions that must be answered and tasks that must be completed. The six domains are related to each other, so decisions made in one of them can have implications on others. For example, choices regarding the secure storage of information on mobile devices (operations domain) will affect the implementation guidelines for mobile applications (applications domain).

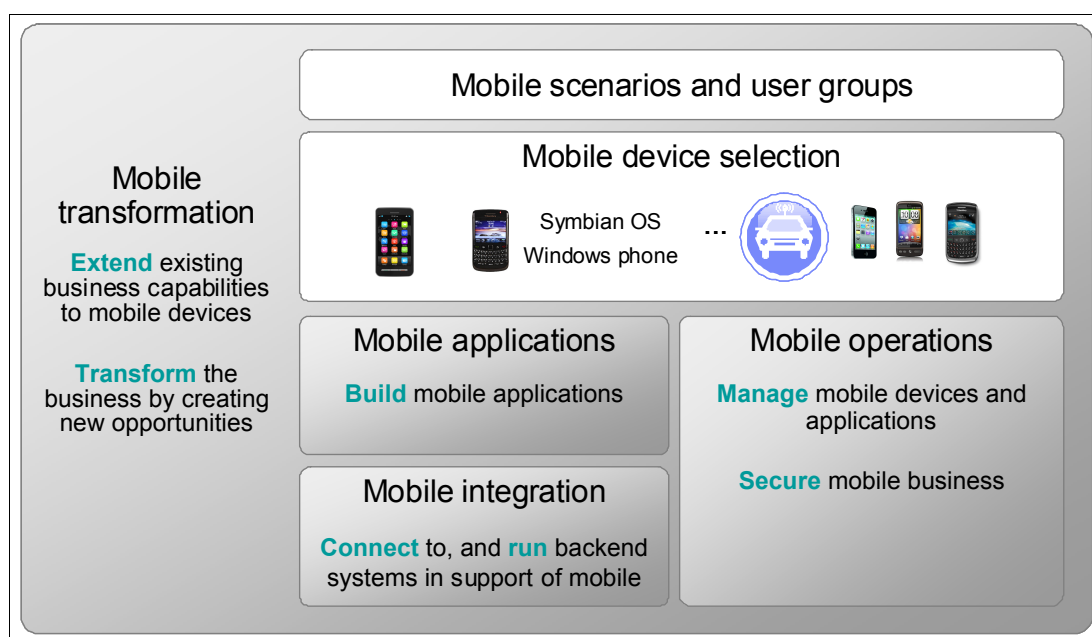


Figure 1-3 Domain model for building a successful mobile strategy

You work through this model using an iterative approach, starting with answering the high-level questions and making high-level decisions. With every iteration you decide additional details. The establishment of governance procedures is critical. As already mentioned, decisions made in one domain can affect other domains, and these impacts must be identified and accounted for within those other domains.

The domain model not only helps with developing a company's initial mobile solution, it helps to speed development of more solutions because you only need to revalidate your earlier assumptions and decisions and make adaptations as necessary.

1.4.1 Mobile transformation

The first task within the *mobile transformation* domain is to define the project's objectives, such as improving one or more key performance indicators. Examples include improving customer satisfaction or attracting more young customers. These targets also provide criteria that can be used to measure the success of the mobile solution later on. The discussion of

objectives and goals is also driven by user feedback and usage analysis of any existing mobile solutions.

The second task is to identify high-level user scenarios that will achieve the defined objectives. These can be completely new scenarios or ones that extend existing applications through a mobile channel. These high-level scenarios will be the basis input for the tasks that must be accomplished in the next domain (mobile scenarios and user groups).

While defining high level scenarios, you must also address the impact of proposed mobile solution on existing products and services offered by the enterprise. A new mobile solution sometimes leads to changes in existing products or even requires changes to the overall business model.

After several scenarios have been identified, create a road map to implement the scenarios. A leading practice is to start with a scenario that is easy to implement and will have a big impact for your business.

When considering the transformation domain, the following questions will help you define why you are entering the mobile space and how you can accomplish the transformation:

- ▶ What goal do I want to achieve?
- ▶ How do I measure the success of the solution?
- ▶ Which scenarios can fulfill the defined goal, at a high level?
- ▶ Does the solution have the potential to alter my underlying business model?
- ▶ What are the short-term, mid-term, and long-term road maps for achieving the mobile strategy?

1.4.2 Mobile scenarios and user groups

The main task of the *mobile scenarios and user groups* domain is adding details to the high-level scenarios that were identified in the transformation domain. You start by determining one or more target user groups for each scenario and then adding the necessary details using the steps shown in Figure 1-4 on page 15.

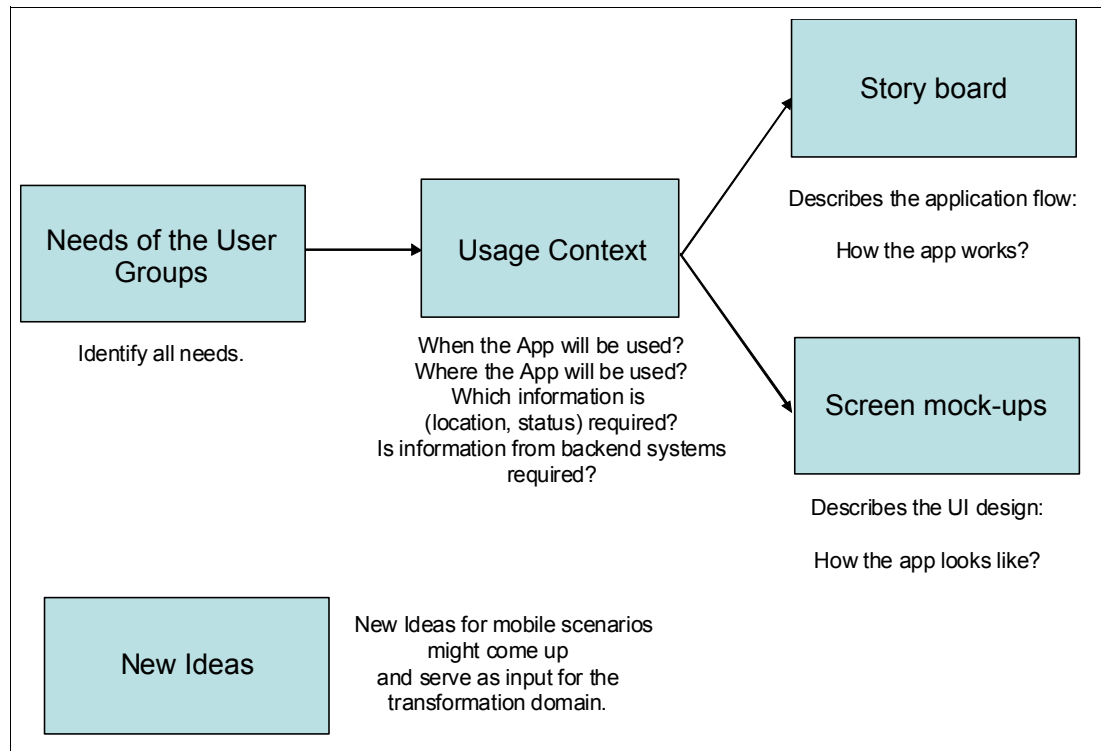


Figure 1-4 Steps to expand a potential mobile scenario with additional details

The first step is to determine how the mobile solution will address the needs and problems of the target users. A good starting point is to consider the user environment, not only in terms of technical conditions such as location and instant messaging status but also in regard to the user's specific needs or problems and why the mobile solution will be used at a specific moment. These considerations can help you focus on the core functionality the mobile solution must provide,

After user needs and environment are determined, the second step is creating a storyboard. A storyboard defines the flow of the application and is typically illustrated by screen mock-ups. The mock-ups show concepts for the main graphical features a user will see at each step of the application.

While detailing specific scenarios, new ideas for additional mobile scenarios might be revealed. This serves as input for further iterative discussions in the transformation domain.

The following questions can help you define the scenario and user group aspects of your mobile strategy:

- ▶ What are the user groups for my scenarios?
- ▶ How does the new mobile solution address the needs of the target user groups?
- ▶ How does each scenario work in detail?

1.4.3 Mobile device selection

The main task in the *mobile device selection* domain is to determine which platforms and devices will be supported by the mobile application. Depending on the user scenario, there might be also special requirements for additional hardware, such as an RFID reader or a bar code scanner. Software requirements must also be considered, including potential encryption mechanisms or centralized management capabilities using a Mobile Device Management

(MDM) system. Any of these considerations might limit the number of supported platforms and devices.

Initially, these issues might seem to be important only for business-to-business or business-to-employee scenarios. However, questions regarding supported platforms and devices are also important in the business-to-consumer environment. In addition to smartphones and tablet PCs, other devices such as televisions, smart home controls, and even motor vehicles can be used as terminals for a mobile solution, and many of these devices rely on mobile platforms as an operating system.

An essential task is to document the decisions you make in choosing your target platforms and devices because the landscape of available devices is evolving rapidly. Detailed documentation will allow you to quickly adapt to future changes. The number of selected platforms and devices also dictates many of the requirements for testing the mobile solution and can affect the selection of the best development approach to follow.

The questions listed here help you to select the appropriate devices for your mobile solution (your decisions might be scenario-specific in that the best answers for Scenario A and Scenario B might differ):

- ▶ Which devices meet the needs of my scenarios?
- ▶ Which devices also meet my non-functional requirements (security, device management, and so on)?
- ▶ Does my solution rely on certain mobile hardware features such as a camera or acceleration sensor?
- ▶ Which devices are my target users already using?
- ▶ For business-to-employee scenarios, can I use existing devices of my employees?

1.4.4 Mobile applications

The *mobile applications* domain focuses on the development of the mobile solution. The solution includes the application with which users interact and the back-end services with which the application communicates. Depending on the enterprise, all or part of this application ecosystem may already be in place or may need to be developed.

One of the first decisions you will face is selecting your development approach. Will it be native, web, or hybrid, as explained in 1.3.3, “Development challenges” on page 9. Each approach has advantages and disadvantages, depending on the requirements of the solution and the supported platforms and devices. A general decision about which approach is best is sometimes not possible, meaning that different mobile solutions might require different development approaches.

The selection of an approach is driven by more than just requirements for access to specific device or platform capabilities. Additional factors include determining if large amounts of data will be stored on the device, and whether the application will have a common user interface and behavior across all platforms or will be adapted to the look and feel of each platform.

The manner in which the application will be distributed to the users is also important. Web applications can be freely distributed over the Internet or corporate intranet without the need for a vendor-specific application store. Hybrid applications and native applications are distributed through vendor-specific application stores or corporate enterprise application stores. Each approach has advantages and disadvantages. For example, web applications (and the web application parts of hybrid applications) can be easily updated in a process you

can control. The update procedure for native applications is controlled by the individual device vendors, so you have to follow their processes.

Software frameworks allow designers and programmers to devote their time to meeting requirements unique to their application rather than dealing with low-level details of providing a working system, thereby reducing overall development time. For example, a team using a web application framework to develop a banking website can focus on writing code particular to banking rather than the mechanics of request handling and state management. Using a mobile JavaScript framework, such as Dojo Mobile and jQuery, can speed development of your mobile solution by providing mobile-specific building blocks for quickly assembling applications using predefined widgets and actions. This allows the developer to spend more time focusing on the application-specific requirements.

Furthermore, you must clearly define which application functions must be run and deployed on the device versus which parts may be centralized and run on the back end. The advantage of running certain functions on the back end is that it can reduce complexity and redundancy in mobile applications across multiple platforms. The disadvantage is that an active data connection is required for the application to function fully.

An important part of the application domain is testing. This includes testing both the back-end services and the application itself, and how the application runs on various mobile devices. Automated testing of the back-end services is widely used. In addition to functional and usability tests, emphasis should be given to load and stress testing. This allows problems to be detected early, before the number of users starts to grow.

For mobile application testing, platform vendors offer a variety of automated tests using simulators or emulators. However, beginning the testing early and doing it directly on the mobile devices is important to reveal any potential performance problems within the expected user experience. As the number of available devices on the market expands, purchasing and operating them becomes expensive. One alternative is to use external testing providers that offer access to a wide range of mobile devices.

Keep in mind that no one approach is suitable for all situations. Within an individual enterprise, there may be different approaches for different applications. The following questions will help you select an appropriate development approach for your scenarios:

- ▶ How does the application access device-specific and platform-specific capabilities?
- ▶ Does the application need to store large amounts of data?
- ▶ Should the application use a common user interface across all platforms or should it use a platform specific UI look and feel?
- ▶ How will the mobile application be distributed to the user?
- ▶ Where will the business logic be located, on the device or at a central location?

1.4.5 Mobile integration

Most mobile applications are not self-contained but instead require access to various back-end services, information, and business processes. The *mobile integration* domain addresses these needs by defining which services and information the mobile application needs to access, whether there are already standardized interfaces to the services and information, and what kind of protocols and security mechanism are required to use the services and information.

In most cases, the recommendation is that mobile applications do not directly access the central system but instead use an integration layer. This integration layer creates a common

interface to be used by all mobile platforms. The diverse needs of the platforms for access to the back-end systems are then handled centrally in the integration layer.

In the enterprise environment, Extensible Markup Language (XML) is the de-facto standard to communicate between systems. In mobile applications, the lightweight and more compact JavaScript Object Notation (JSON) standard is more common. JSON is a lightweight and human-readable communication format that reduces the transfer volume when compared to HTML or XML. The conversion from XML to JSON formats can be done in the integration layer.

As mobile solutions might have a fast-growing user base, you need to think early about how to detect and handle high volume situations. The integration layer must be able to constantly monitor and analyze the number of active users and their behavior. To handle rising user numbers, the integration layer and the enterprise systems must be scalable. Caching can be a potential solution to handle high volumes and peak loads. Caching solutions buffer information that does not change frequently inside the integration layer to reduce the load on the enterprise systems.

The number of active users and usage patterns are also useful in determining which platforms and devices are being used to access the mobile solution. This information is valuable input for the discussion in the device selection domain.

The integration layer provides not only access to back-end services but can also provide access in the other direction by enabling back-end services to send notifications to mobile devices. The integration layer manages the complexity of interacting with the various vendor-specific notification systems for each mobile platform.

The questions shown here will help you to successfully integrate your mobile solution into an existing IT environment by ensuring that you involve every affected party within your company:

- ▶ What information does the application need from existing enterprise systems?
- ▶ How will the application access the data from those existing systems?
- ▶ What are the communication patterns (synchronous versus asynchronous; one-way versus bidirectional)?
- ▶ How does the enterprise detect growing usage of the mobile solution?
- ▶ How can the enterprise handle a potentially fast growing user base?

1.4.6 Mobile operations

The *mobile operations* domain focuses on the secure operation of the mobile solution. Securing a mobile solution has three aspects:

- ▶ Enterprise security
- ▶ Application security
- ▶ Device security

New security policies to support mobile applications might need to be developed within each of these areas.

Enterprise security involves the existing security model imposed by the enterprise for hosting applications and services. This includes authentication, authorization, and accounting (sometimes referred to as AAA), and Secure Sockets Layer (SSL) enforcement.

Policies must be established to describe how information and services can be accessed from a mobile device. It is important to determine if the existing AAA mechanisms are sufficient or if new mechanisms are required. Securing the communication channel to the mobile devices is of great importance also. It might be necessary to prohibit mobile access to certain types of information, which could force changes to be made in the existing back-end infrastructure.

Application security focuses on the client side and includes such matters as the need for a secure connection, secure and encrypted offline storage, application integrity, and user authentication enforcement. Application integrity mechanisms ensure your mobile application is not infected by malware. User authentication policies describe if and how the user must be authenticated to use the mobile application.

Device security refers to the guidelines for managing mobile devices, typically using a mobile device management system. This aspect is essential in business-to-employee and business-to-business scenarios. For consumer scenarios, device management plays a secondary role because a company is not usually in a position, and has no need, to manage the devices used by end users. Bring your own device (BYOD) policies can be a key driver in the discussion of device security. Your mobile strategy must define whether to separate private and business information and applications, and if they are to be separate, how to accomplish that separation. Currently, separation can be handled through difficult and uncomfortable mechanisms such as the so-called sandboxing principle, in which a sandbox (an isolated environment to run applications and store information) allows separation of certain business applications from the private ones.

As part of a BYOD policy, an enterprise must determine what information can be accessed on private devices. Liability issues must be considered, particularly in regard to situations when a private device is lost or damaged while being used for business purposes. The policy must define whether private devices are managed (and therefore controllable) by the company's mobile device management system. These policies should be reconciled with co-determination parties such as worker councils or unions, and the legal department if applicable in your country.

Another aspect of the mobile operations domain is application distribution. You must define where and how to publish the mobile application, such as through vendor-specific application stores or a private enterprise applications store, and how to update the mobile application within the store. The frequency of updates and the targeted users (business, employee or consumer) will help to determine the appropriate approach. For example, using a hybrid application allows for partial application updates being distributed using a vendor-specific application store, without the requirement to go through a potentially time-consuming vendor-specific approval process.

User support is another important consideration when putting a mobile solution into operation. Increasing user numbers likely imply additional support requests. So consider which support of these mechanisms you will offer:

- ▶ User self-support with documentation
- ▶ Email support to a support team
- ▶ Call center support

The overall strategy behind the mobile solution must account for the effort and time involved in training the support staff to handle questions. It also must consider any additional hardware and software requirements related to the support staff.

The following questions will help you determine how to run and manage your mobile solution once it is implemented:

- ▶ What security mechanisms are needed to secure access to device, the information on the device, and the communication channel to the device?
- ▶ Are additional security mechanisms needed on the enterprise IT side?
- ▶ How will the mobile applications be managed?
- ▶ How will the mobile solution be supported?

1.4.7 Finalizing your mobile strategy

Now that you have defined your mobile strategy, consider using a Mobile Application Platform to help you bring your mobile solutions online more quickly.

A Mobile Application Platform is used to develop and operate mobile solutions and is a long-term approach to managing a variety of mobile solutions, not only a single application. The main driver for adopting these platforms is the diversity in the mobile market:

- ▶ Device diversity: Numerous smartphones and tablets
- ▶ Platform diversity: Android, BlackBerry, iOS, Windows Phone, and so on
- ▶ Network diversity: Low and high bandwidth, online and offline
- ▶ User group diversity: Employees, business partners, consumers

With Mobile Application Platforms, you can develop and run cross-platform applications capable of running on several mobile operating systems. Mobile Application Platforms also offer features to connect to existing enterprise services and information sources, and allow you to develop and operate services retrieving and transforming information from enterprise systems. They are also able to manage mobile applications centrally (deploy, update, withdraw). In addition, Mobile Application Platforms allow you to manage mobile devices or can be easily integrated with sophisticated Mobile Device Management (MDM) systems. They offer an application store to browse and download mobile applications.

There are three types of Mobile Application Platforms:

- ▶ Mobile Enterprise Application Platforms (MEAPs): These address the specific needs of mobile applications in the business-to-employee environment such as secure access through a VPN. They provide sophisticated mobile device management and an enterprise-specific application store to distribute applications.
- ▶ Mobile Consumer Application Platforms (MCAPs): These address the specific needs of mobile applications in the business-to-consumer environment such as quickly adapting to a volatile customer base. A sophisticated mobile device management system is not required, because an enterprise does not typically manage mobile devices of its customers. A specific application store is usually not required; the vendor-specific application stores are used instead.
- ▶ Mixed Mobile Application Platforms (MMAPs): These address the needs of mobile applications for both employee-facing and consumer-facing environments.



Exploring the solution architecture

This chapter introduces the IBM MobileFirst portfolio and related products. It describes how the mobile applications, mobile integration, and mobile operations domains are mapped to different tasks within the IBM MobileFirst portfolio. Finally, various user group scenarios are described to illustrate how IBM mobile solutions handle various requirements.

The following topics are covered:

- ▶ 2.1, “Understanding the concepts behind IBM MobileFirst” on page 22
- ▶ 2.2, “Addressing mobile challenges” on page 25
- ▶ 2.3, “Understanding the products used in the solution” on page 32
- ▶ 2.4, “Adapting your mobile solution to a different audience” on page 43

2.1 Understanding the concepts behind IBM MobileFirst

IBM MobileFirst is a portfolio of mobile solutions that combines the key elements of a mobile application platform with the management, security, and analytics capabilities needed for an enterprise. In addition to meeting mobile-specific requirements, the portfolio provides for rapid integration between social and cloud services and also back-end technologies that help secure and manage strategic business processes.

IBM MobileFirst portfolio splits the enablement of your mobile strategy into these logical task groups:

- ▶ Building and connecting mobile applications (IBM MobileFirst platform)
- ▶ Managing mobile devices and applications (IBM MobileFirst management)
- ▶ Securing enterprise data and services (IBM MobileFirst security)
- ▶ Analyzing mobile interactions to improve responsiveness and return on investment (IBM MobileFirst analytics)

The IBM MobileFirst architecture is open and extensible by design, enabling organizations to choose the solution that integrates with their existing environment and solves their most pressing business needs. The mobile solution developed in this book includes the following products:

- ▶ IBM Worklight to develop, run, build, and manage mobile applications
- ▶ IBM WebSphere CastIron Hypervisor Enterprise Edition to integrate back-end and cloud services
- ▶ IBM WebSphere DataPower® appliances to provide application integration capabilities

2.1.1 Building mobile applications

As outlined in 1.4.4, “Mobile applications” on page 16, mobile application development differs from traditional application development because of recent trends such as IT consumerization, demands for faster time-to-market, and higher expectations regarding the user experience, to name a few. Additionally, the fragmentation among mobile devices and mobile operating systems has limited the notion of *write once, run anywhere* that many developers followed with programming languages such as Java. Whether the variation between mobile devices is the screen size, screen resolution, or even the JavaScript engine embedded in the mobile device's browser, the need to develop and test applications on an ever-expanding list of devices will continue. And although none of these variations is specific to mobile, the pace at which these variations are being introduced is creating a constant need to release new and better applications to satisfy growing consumer demand.

Agile software development is a methodology that enables planning, development, testing, and delivery of software in a way that encourages rapid and flexible response to change. Agile methodologies are not unique to mobile, but they match well with the rapid pace at which users expect new applications to be developed and released. Using an agile software development approach, a mobile application development project can rapidly produce new releases that contain small, incremental feature enhancements, with quality maintained through iterative testing.

One example of an agile development framework is SCRUM, in which development and testing is done within an iterative, timeboxed period called a *sprint*. By using the SCRUM methodology, shown in Figure 2-1 on page 23, a small set of features can go through the complete development lifecycle, including planning, implementation, and quality assurance, and be delivered to users rapidly. The review at the end of each sprint can include user

feedback that then feeds into the planning of the next sprint, so improvements can be based on direct user input.

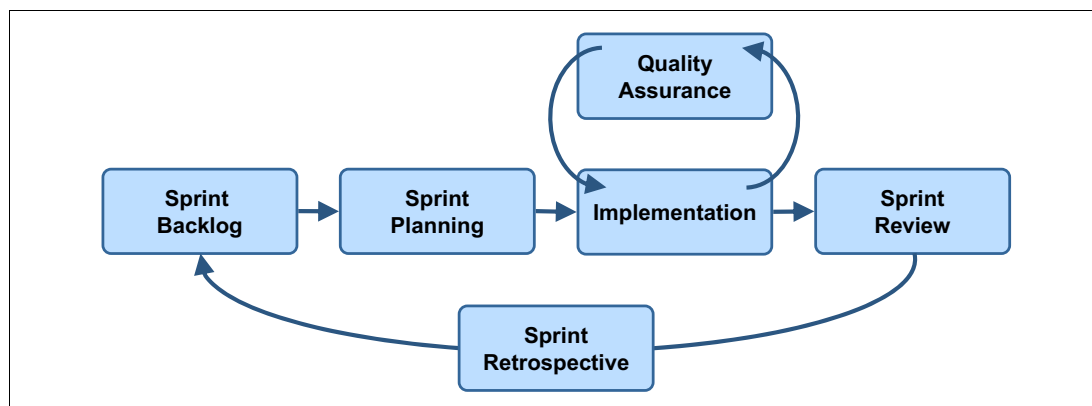


Figure 2-1 Steps within a SCRUM methodology sprint

Implementing a mobile strategy extends far beyond just the developers who write the application code. A roadmap for achieving the strategy must be defined with specific goals that can be shared with the team. The work roles necessary to implement the roadmap must be defined to ensure proper staffing. And if the mobile application needs connectivity to enterprise data, the list of required back-end services and how these services will be utilized must be determined.

During application development, the IBM Worklight separates the roles of native developer, hybrid developer, and web developer through its tooling. This helps to split the workload when a broad set of requirements requires expertise in UI, device capabilities, application logic, and so on. For example, if the application will be using a relatively new technology, such as near field communication (NFC), certain device capabilities will be implemented differently based on the specific device and operating system. In these cases, a native developer can implement a module that exposes part or all of the new functionality to an abstraction layer such as an Apache Cordova plug-in. Creating an abstraction layer helps a web developer access unique device capabilities and incorporate these features into the user interface without needing to know the specifics of the device or its implementation.

Actively seeking feedback about a new application or version through a pilot or beta group is a good way to ensure that your initial release does not fail and cause negative perceptions of your company. This is especially true for the first release of an application, when users do not have the option of falling back to an earlier, more stable version. Gathering feedback is made easier by using an application store such as IBM Worklight Application Center, which enables you to offer a prerelease version of an application to a limited set of users. You can then analyze the feedback in a *sprint retrospective* or lessons-learned session to identify what was good and bad about the release.

If you are developing your organization's first mobile application, valuable early planning information can be found in *A mobile application development primer*, an IBM white paper that is available at the following address:

<http://public.dhe.ibm.com/common/ssi/ecm/en/raw14302usen/RAW14302USEN.PDF>

The IBM Mobile Development Lifecycle Solution also can help with planning for enterprise grade mobile application development projects, including best practices, storyboarding, requirements capturing, and agile development methodologies. With this platform you can optimize device testing by distributing the work to specialists in each type of device.

Learn more about the IBM Mobile Development Lifecycle Solution at this address:

<http://www.ibm.com/software/products/us/en/imdl1s/>

2.1.2 Connecting and running mobile systems

Today, only a small portion of mobile applications run strictly offline with no connection to a back-end system. Most mobile applications connect to new or existing enterprise services to provide critical user-related functions. For instance, a mobile application can allow shoppers to shop anywhere, at any time, independent of the a store's operating hours. But their orders must still be processed using the store's existing e-commerce platform. IBM Worklight helps ease the integration and communication with back-end services.

Push notifications are a way for a mobile application to send information to a mobile device even when the application is not being used, such as when a news application alerts you to a breaking news story. IBM Worklight includes a unified notification framework (Figure 2-2) that offers a consistent mechanism for such pushes. Because each mobile platform has a different mechanism for these notifications, this unified framework allows a developer to send the push notifications without having to know the details of each targeted device or platform.

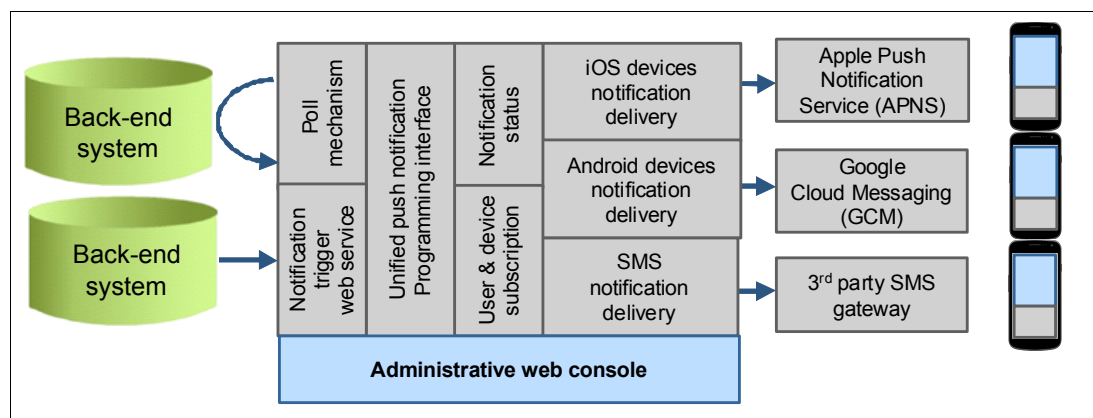


Figure 2-2 Unified push notification architecture in IBM Worklight

In terms of connectivity, mobile applications can operate offline, online, or in a mixed mode. IBM Worklight uses a client/server architecture that can detect not only whether a device has network connectivity, but also the quality of the connection. Acting as a client, your mobile application periodically attempts to connect to the server and to assess the strength of the connection.

An offline-enabled mobile application can be used when a mobile device lacks connectivity, but some functions may be limited. So when creating an offline-enabled mobile application, it is useful to store information on the mobile device that can help preserve its functionality in offline mode. This information is typically from a back-end system (enterprise resource management, warehouse management, business intelligence, and so on), which means you must consider back-end data synchronization as part of the application architecture. IBM Worklight includes a data synchronization feature called the JSON store, which provides a pattern for data exchange, as shown in Figure 2-3 on page 25. This pattern contains a set of operations to create, read, update, and delete data records from a data source. Each operation is queued when operating offline. When a connection becomes available, each operation is transferred to the server and performed against the source data.

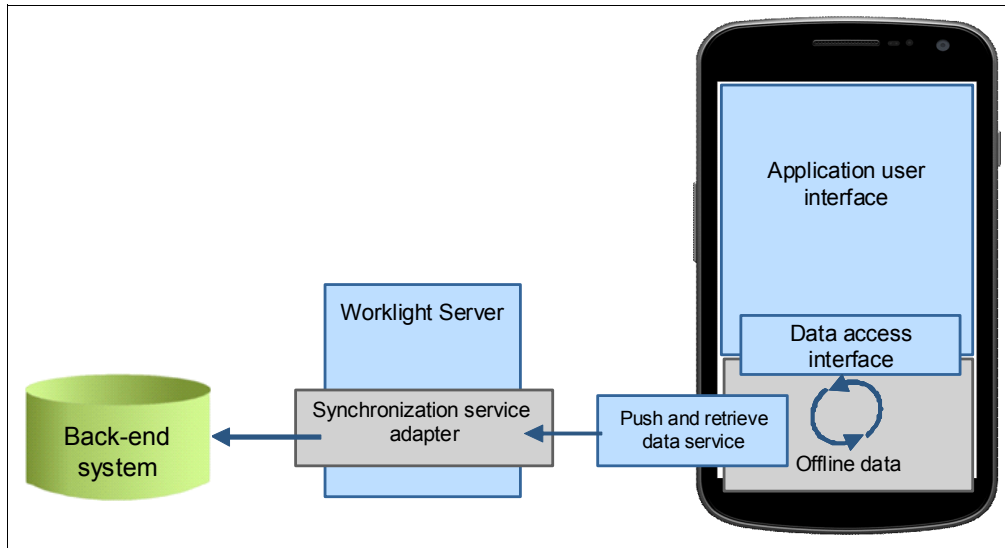


Figure 2-3 Offline data access and data synchronization using a JSON store

In addition, IBM Worklight simplifies version management and mobile application compatibility. When a user starts a mobile application, the application communicates with a server. Using this server, IBM Worklight can determine if a newer version of the application is available, and if so, tell the user about it or automatically push an application update to the device. The server can also force an upgrade to the latest version of an application to prevent continued use of an outdated version.

2.2 Addressing mobile challenges

IBM Worklight provides many capabilities that address some of the specific challenges described in 1.3, “Mobile challenges” on page 7. In this section, you learn how IBM Worklight addresses each of these challenges.

2.2.1 Business challenges

Many business opportunities and challenges cannot be solved with *prepackaged software*. Prepackaged software is designed to solve a particular general business need, with limited flexibility in terms of customizing the application to satisfy every individual business case. In contrast, a customizable solution such as IBM Worklight provides tools to help you achieve specific business targets, such as improving customer relationships, reducing paperwork and paper usage, or intensifying the marketing of certain products.

IBM Worklight provides a single platform for the mobile communication channel. With this platform, you can develop and deliver a mobile application and then adapt it as business requirements change. For example, IBM Worklight provides a Direct Update feature to assist in limiting the exposure caused by frequent releases of application updates. With Direct Update, you can actively push urgent mobile application updates to your user base.

One of the latest industry trends is the proliferation of mobile devices and the pace in which these devices go to market. Businesses must have the agility to respond to the release of multiple new devices each year, each with new or updated operating systems, different screen sizes, and so on. To provide this agility, IBM Worklight leverages open standards, such as HTML, CSS, and JavaScript, and solutions such as Apache Cordova, Eclipse Foundation,

and the Android and Apple SDKs. This provides flexibility in implementing your mobile communication channel and can reduce the time-to-market for a new version of an application that supports a newly released device.

In addition to embracing open standards, IBM Worklight and IBM WebSphere Cast Iron give you a set of tools to monetize the reuse of your service-oriented architecture (SOA) services. The reuse of services, which has been a common practice in the IT industry for many years, can be extended to businesses services also. This means you can publish the consumable parts of your business models and other mobile applications can reuse the functionality. By having a Web API Marketplace to host your applications (see WebSphere Cast Iron Web API), consumers of your services can easily locate and consume these services in their own applications. For instance, a public transportation company could publish routing and ticket services as part of its own mobile solution, and then a separate company could create a mobile application to optimize the travel experience and, as part of that application, could reuse the services published by the transportation company.

With IBM Worklight, your company can focus on reducing the effective distance between you and your customers and focus on engaging your user base. The ability to reach customers at all times, wherever they are, improves and accelerates communication. For instance, a mobile application can help to establish new sales channels or improve public awareness of your company. This concept is often referred to as *systems of engagement*. Systems of engagement refers to the transition from current enterprise systems, designed around discrete pieces of information (records), to systems such as databases and mainframes, which are more decentralized and incorporate technologies that encourage peer interactions. As customers become more engaged in communicating with the company, the company can use what it learns to make sure that its future products and services are as effective as possible in solving the true needs of its customers.

2.2.2 Concept and design challenges

The user experience and design of your mobile solution expresses your corporate identity and ultimately influences your brand value. When comparing notebooks, smartphones, and tablets, the most obvious differences are the device shape and hardware specifications. The mouse and keyboard, used for traditional systems such as desktop and notebook PCs, have been replaced on mobile devices by touch screens and small, device-based keyboards. It is important to design a mobile application to account for these differences. An application that is easy to use with a mouse and full keyboard on a large screen may not be usable on a mobile device.

The *form factor* of a device refers to the size and shape of the screen and defines the available visual representation space that your application can use on the device. Because your mobile audience may use devices with different form factors and keyboards, your concept and design must address these differences. Defining company-wide mobile user interface guidelines can assist in ensuring that good user interface design is applied to all applications. These guidelines must not focus on just a small subset of mobile platforms such as iOS and Android. Although an initial release of an application might be limited to only a few platforms, you will likely expand your application to other platforms eventually, and these platforms can have different navigational or other requirements. If your company believes in a strong brand identity, your design guidelines should reflect your brand across all of the mobile platforms you use.

In developing these guidelines, your design considerations can include the following items:

- ▶ Responsive layout, where what is shown on the screen changes based on the device's screen size
- ▶ Screen flow and transitions
- ▶ Selective display of content, showing mandatory content on smaller screens and adding optional content on larger screens
- ▶ Input assistance, including auto-complete and voice recognition
- ▶ Animations and visualizations to convey information using graphics instead of text

The guidelines might also cover how applications can benefit from certain mobile device capabilities to simplify data input:

- ▶ Selecting a destination address from the user's address book
- ▶ Using location services to establish the location of the device
- ▶ Accepting user input through voice recognition tools
- ▶ Using the camera to scan a bar code

More information about leading design practices for applications on multiple device platforms is available from IBM developerWorks® at this address:

<http://www.ibm.com/developerworks/web/library/wa-interface/index.html>

New mobile devices are released frequently, and as they emerge, the concept and design of your mobile application must adapt. IBM Worklight supports the detection of new mobile devices and essential feature indicators using a concept called *skinning*. Skinning is the ability to change the look and feel of software by applying a *skin*. Some skins change only the look and feel of an application; others go further and rearrange the on-screen elements so that the application is easier to use.

As a result of new mobile devices being introduced at a near record pace, the device fragmentation causes developers to have more difficulty writing true cross-platform mobile applications. Many developers now focus on new design patterns, such as responsive design, where the focus is more on the application content than on how that content is displayed on the screen. By using a responsive design approach, splitting the functionality into self-contained visual elements, it becomes easier to adapt for new types of mobile devices. For example, if a new device with a square screen is released, you can reuse existing visual elements and rearrange them to fit the new screen. Responsive design relies on certain constraints, such as the availability of a feature, and then outlines how your application responds to a particular screen size or orientation.

One of the key elements of responsive design is the use of CSS *media queries*. Media queries were added in CSS3 and provide a rich expression language coupled with traditional CSS rules, an approach that allows developers to tailor the look and feel of an application using style sheets based on specific page characteristics, such as width and height.

2.2.3 Development challenges

Choosing the correct development approach depends heavily on the specifics of your mobile application and its functional requirements. So mapping your requirements to an appropriate development approach is the first step in a mobile development project.

Table 2-1 outlines the major aspects of the four development approaches discussed in 1.3.3, “Development challenges” on page 9. Reviewing this list can help you choose which development approach is correct for your particular mobile application.

Table 2-1 Comparison of mobile development approaches

Aspect	Web development	Hybrid development	Hybrid mixed development	Native development
Learning curve	Easy	Medium	Medium	Hard
Application performance	Slow	Moderate	Moderate	Fast
Required device knowledge	None	Some	Some	A lot
Development lifecycle (build/test/deploy)	Short	Medium	Medium	Long
Application portability to other platforms	High	High	Medium	None
Support for native device functionality	Some	Most	All	All
Distribution with built-in mechanisms	No	Yes	Yes	Yes
Ability to write extensions to device capabilities	No	Yes	Yes	Yes

With IBM Worklight, you can deliver your first application using any of these approaches, enabling you to chose between web, hybrid, hybrid mixed, or native application development.

If your organization already has a web development team, a web-based or hybrid development approach might be the easiest. This approach is also ideal if your application developers have not yet written native applications for mobile devices, in which case they must first learn about the different devices the application will support. This learning curve can cause delays and put the project at risk, so choosing a web or hybrid approach, which reduces the need for new skills, might be the best option.

Each approach has advantages and disadvantages based on factors such as development lifecycle time, portability to other platforms, native functionality, and the ability to write extensions to device capabilities. Your choice depends not only on your team’s existing development skills and device knowledge, but also on your specific mobile application requirements.

IBM Worklight enables you to develop your mobile application cohesively. This means you can develop your mobile application and your mobile services using common tools and a common implementation. To move your mobile application to a specific platform, you simply adjust the common implementation logic as needed.

Security can be added in a modular way, so developers can add or enable different security mechanisms, such as authentication, after the initial logic and user interface have been implemented. Because IBM Worklight transparently encrypts the communication between your mobile application and the mobile integration server, your application starts with this minimum level of security already built in.

2.2.4 Infrastructure challenges

General mobile infrastructure challenges are described in 1.3.4, “Infrastructure challenges” on page 10. But these challenges are unique for each company because each company has different quality of service requirements regarding application availability, scalability, and communication. The components of the IBM Worklight allow you to make your mobile applications highly available and scalable, and provides adapter technology that allows your mobile applications to access your existing business services, reducing resource requirements.

Placing IBM Worklight on proven technology, such as WebSphere Application Server clusters, can help you achieve your required levels of service of availability and scalability. In addition, a load balancer can be put in front of the clustered solution to improve performance, and a security gateway can be placed there to enable the use of a separate security infrastructure (see 2.3.3, “IBM WebSphere DataPower appliances” on page 41). And the use of elastic caching mechanisms can help ensure proper management of peak loads (see 2.3.4, “IBM WebSphere eXtreme Scale and the IBM WebSphere DataPower XC10 caching appliance” on page 42).

IBM Worklight Server provides adapters that can issue requests to web services, databases, and other applications on behalf of mobile applications. These adapters can be used to combine information from multiple sources into a single response to the application. The adapters also can use server-side JavaScript to modify data before the request and after the response. This can be used to filter data returned from the enterprise service that is not needed by the mobile device and limits the amount of data transferred to the mobile device. If a full caching mechanism is not in place, the adapter can also cache frequently requested data.

IBM Worklight adapters can communicate with the back-end systems using a variety of protocols:

- ▶ REST or SOAP over HTTP for web services
- ▶ JDBC for databases
- ▶ JMS for a message queue
- ▶ WebSphere Cast Iron for various on-premises and cloud services

2.2.5 Security challenges

IBM Worklight addresses the security challenges described in 1.3.5, “Security challenges” on page 11 on different levels in terms of users, mobile devices, the network, the mobile middleware, and systems integration. It does this by following these simple principles:

- ▶ Always identify the user of a mobile device.
- ▶ Always secure your client application and data.
- ▶ Use only secure communication.
- ▶ Always build upon proven middleware security technologies.

Always identify the user of a mobile device

IBM Worklight uses the concept of security layers. A mobile device consists mainly of three stacks: the mobile operating system, the mobile application sandbox, and the mobile application. If an operation inside your mobile application is intended to only be available to certain users, then you must ensure that the user of the mobile device from which the request originates is one of those permitted users.

Additional security concerns arise with *rooted devices*, in which the user has administrative rights to make changes at the operating system level. This allows skilled users to self-identify themselves as *any* user. So, a mobile application cannot trust a rooted device for user identification and must instead use its own security mechanism.

IBM Worklight provides tools and middleware for user authentication and authorization. Two features, in particular, can help identify the current user of a mobile device:

- ▶ **Offline authentication**
With offline authentication, a mobile application on a device that is not connected to the server can use the device runtime components of IBM Worklight to authenticate the user with a local challenge mechanism, such as asking a question that only the authorized user will be able to answer.
- ▶ **Authentication integration framework**
The framework significantly simplifies security constraints. For example, a mobile application can be set to require a log-in sequence every time an application starts and before any further functions can be accessed.

Always secure your client application and data

Experts know that it is only a matter of time, resources, and criminal determination before any security mechanism can be overcome. And when overcome, the compromised mechanism can be exploited to take control of the device or gain unauthorized access to sensitive information stored on it.

IBM Worklight creates a unique identifier for each device the first time the device connects to the server, based on device information such as the device's identifier, OS version, and device type. To address potential threats, the system administrator can use this unique identifier to disable the company's applications on devices reported as stolen or breached. The same mechanism can be used to disable applications from running on devices that do not meet your security standards.

IBM Worklight has several features to help you implement security within a mobile application:

- ▶ **Authenticity testing**
Before a mobile application calls a service, you can perform authenticity testing to determine if the application has been compromised. IBM Worklight transparently checks to determine if the web resources of a hybrid application have been modified and, if so, disables the application's access to back-end systems.
- ▶ **On-device data protection**
Mobile devices usually store websites, images, and data files directly on their file system, so anyone with sufficient rights to a device can access your application data and logic. To counteract this, IBM Worklight provides on-device data protection that encrypts application data and logic on the device. In this way, the mobile operating system has no access to the images, application business logic, or user-specific application data on the device.
- ▶ **Encrypted on-device storage**
This is local, encrypted storage that resides on the mobile device and requires a password to be accessed by an application. Instead of using file-based solutions with individual data access methods, developers can use encrypted on-device storage through a simple interface provided by IBM Worklight.
- ▶ **Source code obfuscation**
If your application logic uses an interpreted programming language, the source code is readable by anyone who can access it. When IBM Worklight is used to develop hybrid

applications, its tooling automatically obfuscates all web source code and resources at build time, making it difficult for humans to understand even if they can access it. This is important, because code that is readable is more easily hacked than code that is not readable.

Use only secure communication

IBM Worklight encourages the use of trusted entities. Declaring trusted network sources and using certificates to establish trusted entities enables your mobile application to clearly identify which sources are trustworthy and which are not. IBM Worklight transparently encrypts all communication between a mobile application and its middleware layer using Secure Sockets Layer (SSL) technology. In addition, a secondary security method, such as checksum, is used to provide an additional layer of security.

When a Worklight application first runs on a mobile device, it creates a pair of public-key infrastructure (PKI) keys. It then uses these keys to create a secondary encryption key that is based on unique information from the device and application, and sends them to the Worklight server for authentication purposes. But a key pair alone is not sufficient to sign public characteristics because any application can create a key pair. For a key pair to be trusted, it must be signed by an external trusted authority, creating a certificate. The process of obtaining such a certificate is called provisioning. A possibility with IBM Worklight is to create a certificate for an entire device, including all of its applications, instead of having to reissue a certificate for each application.

Always build upon proven middleware security technologies

High security standards must be applied to mobile middleware because the middleware handles all communication between a mobile device and the company's back-end services. And with secure communication established, you can focus your efforts on your mobile application content.

The security framework of IBM Worklight provides a basic set of security standards for mobile applications and integrates seamlessly with a variety of external security solutions. For further information regarding the security framework in IBM Worklight, see *Improve your mobile application security with IBM Worklight*, an IBM white paper:

<http://public.dhe.ibm.com/common/ssi/ecm/en/wsw14198usen/WSW14198USEN.PDF>

Unlike web applications, mobile applications are not instantly updated when new versions are available. This means that new security updates are not always immediately installed on all devices. To answer this need, IBM Worklight provides enterprise application version distribution and application version management. These services allow your company to reduce maintenance efforts and more easily deploy new versions of your mobile application. Administrators can even remotely disable old versions of a mobile application when an outdated version is unusable or poses a potential security threat to users or systems.

2.3 Understanding the products used in the solution

The following sections describe the capabilities of the products used to develop the real-world scenario that are in this Redbooks publication.

2.3.1 IBM Worklight

IBM Worklight is a mobile application platform containing all of the tools needed to develop a mobile application, including implementation, testing, and distribution. It is available in two editions:

- ▶ IBM Worklight Enterprise Edition
- ▶ IBM Worklight Consumer Edition

The consumer and enterprise editions are identical except for their licensing models; the consumer edition is licensed per mobile application although the enterprise version is licensed per device.

IBM Worklight facilitates the creation and delivery of rich, secure mobile applications. It simplifies end-to-end security and service integration between mobile applications and back-end systems. It makes cross-platform application development easier through the use of standards-based technologies such as HTML5 or JavaScript, and tools that can access native device capabilities. And it enables the complete spectrum of mobile application development approaches listed in 1.3.3, “Development challenges” on page 9, from pure web-based mobile applications to pure native mobile applications.

IBM Worklight consists of five core components, as shown in Figure 2-4.

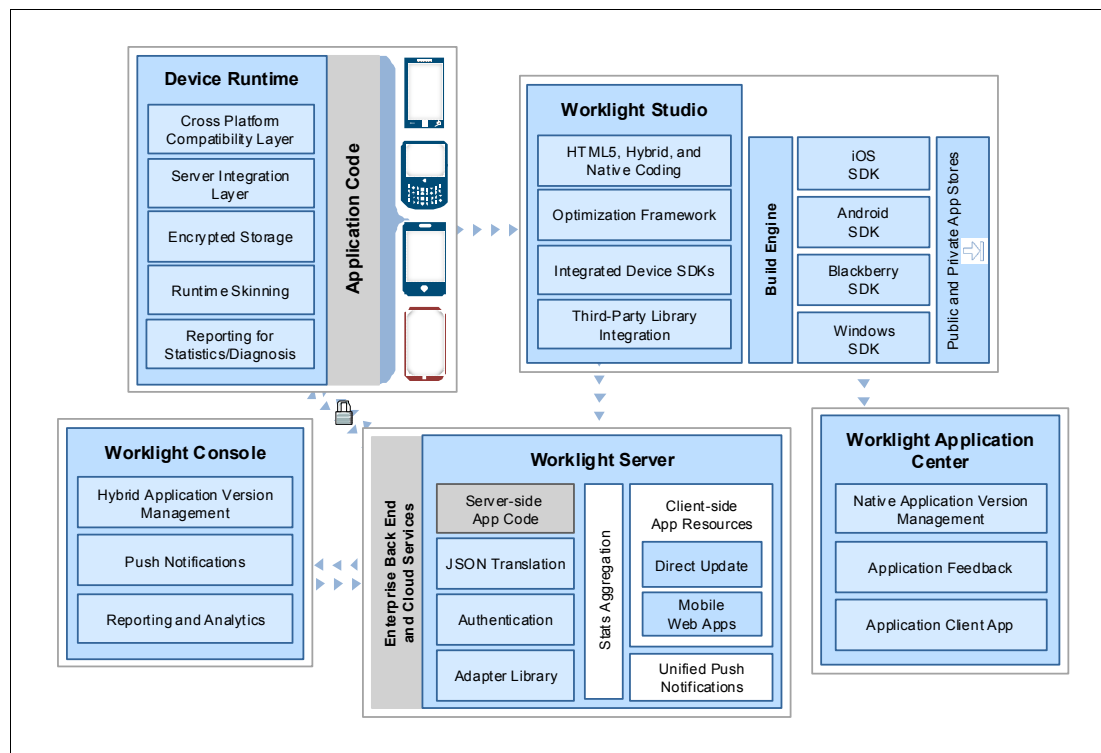


Figure 2-4 The five components of IBM Worklight

Each component provides a specific set of functions and supports a different stage in the lifecycle of a mobile application:

- ▶ **Worklight Studio:** An Eclipse-based development environment with a rich set of tools for developing cross-platform applications
- ▶ **Worklight Device Runtime:** A runtime environment with consistent APIs that can work across different mobile platforms to access native device functionality and integrate with existing services and security
- ▶ **Worklight Server:** A runtime server that enables secure access to enterprise services, application version management, unified push notifications, and direct application updates
- ▶ **Worklight Console:** A web-based user interface for real-time analytics, push notification authoring, and mobile application version management
- ▶ **Worklight Application Center:** A private, cross-platform mobile application store that is tailored for the specific needs of an application development team

Worklight Studio

Worklight Studio is an Eclipse-based integrated development environment (IDE) that provides a set of tools to create a fully operational mobile application for various mobile operating systems, and to integrate applications with existing services. By enhancing Eclipse with custom plug-ins, IBM Worklight enables developers to use a single IDE to build all of your enterprise applications, from server applications to applications for different mobile device operating systems.

Developers get some powerful capabilities:

- ▶ **Pure native development and web development technologies**
IBM Worklight Studio supports pure native development and web development technologies such as HTML5, Apache Cordova, and Java. You can develop mobile applications with pure HTML5 or use a compatible JavaScript framework, such as jQuery Mobile, Dojo, or Sencha Touch, to simplify development with the reusable user interface widgets and commonly used functions that are provided by the frameworks. You can use Apache Cordova to enable your mobile application to access native device functionality. And if you need to access a special device module, such as one for near field communication (NFC), you can develop a native extension that can be exposed to JavaScript through an Apache Cordova plug-in (a small native-to-JavaScript wrapper).
- ▶ **Shell development**
For hybrid mobile applications, IBM Worklight uses a default hybrid shell that offers all of the capabilities necessary to use web and native technologies. Shell development enables your organization to separate native component implementations from web-based implementations and to split this work between different developers. For instance, by creating a custom shell, you can add third-party native libraries, implement custom security, or provide extended features specific to your enterprise. Shells can also be used to restrict or enforce specific corporate guidelines, such as design or security rules. For example, a shell can be used to add a default style to your mobile application, or to disable the device's camera.
- ▶ **Optimization framework**
With IBM Worklight Studio, a common environment is used as a basic development point and all environments can share the same base code. Then, to create an optimized version for an Apple iPad tablet, for instance, you can create a variant of the base and implement only the required changes. An optimization framework called *runtime skinning* enables your mobile application to switch at run time between different sets of customizations.

- Integration of device-specific SDKs

Each vendor of mobile devices supplies its own development environment as part of a software development kit (SDK). IBM Worklight Studio generates a project for each supported SDK (for example, Xcode for iOS development). Some vendors require you to use their SDK for specific tasks, such as building the binary application. The integration of device-specific SDKs within IBM Worklight Studio links your Worklight Studio project with the native development environment (such as Xcode), enabling the developer to seamlessly switch between a native development environment and IBM Worklight Studio. Any change in the native development environment is mirrored back to your IBM Worklight Studio project, reducing the need for manual copying.

- Third-party library integration

Depending on your programming approach, your mobile application can include several JavaScript frameworks, such as Sencha Touch, jQuery, or Dojo. This third-party library integration promotes code reuse and reduces implementation times. Many companies underestimate the effort involved in developing a framework because testing efforts for quality assurance are not fully taken into account. In the case of a shell project, different kinds of compatible native code or libraries can be included.

- Integrated build engine

The build chain of IBM Worklight Studio combines common implementation code (used on all target platforms) with platform-unique code that is used on specific target platforms. At build-time, the integrated build engine of Worklight Studio combines these two implementations into a complete mobile application. This reduces IT development costs by using a single, common implementation for as much of the mobile application as possible, instead of a unique implementation for every supported platform.

- Integrated development tools

By extending the Eclipse IDE with custom plug-ins, Worklight Studio can be used to develop all components of your application, including the mobile application and integration modules (called IBM Worklight adapters) from within the same development environment. Integrated development tools allow you to develop and test these adapters seamlessly within IBM Worklight Studio.

- Mobile browser simulator

IBM Worklight Studio offers a mobile browser simulator that can be used during the development cycle to test mobile web and hybrid applications using a desktop browser. Many of today's desktop and mobile browsers are based upon the WebKit engine as the underlying core technology, providing a common platform for developing applications that support HTML5, CSS3, and JavaScript. This consistent browser engine allows developers who host their mobile browser simulator on a WebKit-based desktop browser, such as Chrome or Safari, can validate the application's behavior prior to deploying it on the actual device. When the developer is ready to test on the actual device or mobile emulator, they can verify that the core WebKit engine resident on many mobile devices, including Android and iOS, provide the same consistent user experience that was verified when testing using the browser. In addition to supporting cross-platform browser testing for mobile devices, the mobile browser simulator also provides implementations of the various Apache Cordova APIs. By providing default implementations of these Cordova APIs, users can test hybrid applications that use device features without having to run on the actual device, decreasing development time and effort that would be required to repeatedly deploy the application to a device.

IBM Worklight Studio is available in three editions. The Developer Edition provides all of the tools needed to build a mobile application. The Consumer Edition and Enterprise Edition add enterprise-level security and integration with the IBM Application Center store.

Because it is based on Eclipse, Worklight Studio gives you all of the flexibility and extensibility that Eclipse provides, such as adding new functionality with plug-ins. For instance, an IBM Rational® Team Concert plug-in can be used to control your source code, track changes, and create daily builds without installing an additional development application.

IBM Worklight Studio also provides a set of Ant tasks that can be used to run a mobile application build for various platforms. You can distribute build tasks to a variety of build machines running Apple OS X (for an Apple iOS binary) or Microsoft Windows (for a Microsoft Windows Phone 8 binary), among others. Using this mechanism can reduce IT resource requirements because developers no longer need access to multiple build machines to create binaries for specific mobile platforms.

Worklight Device Runtime

IBM Worklight Device Runtime ensures that your mobile application has access to a variety of IBM Worklight features during run time. Its components reside on the mobile device and consist of libraries that are bundled into the application to enable additional runtime capabilities. The libraries integrate your mobile application with IBM Worklight Server using predefined communication interfaces and simplify application development by providing unified access to native device functionality.

The native, hybrid, mixed hybrid, or web-based application programming interfaces (APIs) in IBM Worklight Device Runtime provide support for all mobile development approaches with enhanced security and integration features. For hybrid and mixed hybrid applications, the included Apache Cordova plug-ins add native capabilities and cross-platform user interface controls.

IBM Worklight Device Runtime components deliver a uniform bridge between web technologies (HTML5, CSS3, JavaScript) and the native functions available on various mobile platforms. Figure 2-5 shows how the architecture of IBM Worklight Device Runtime combines native and web technology to drive mobile development.

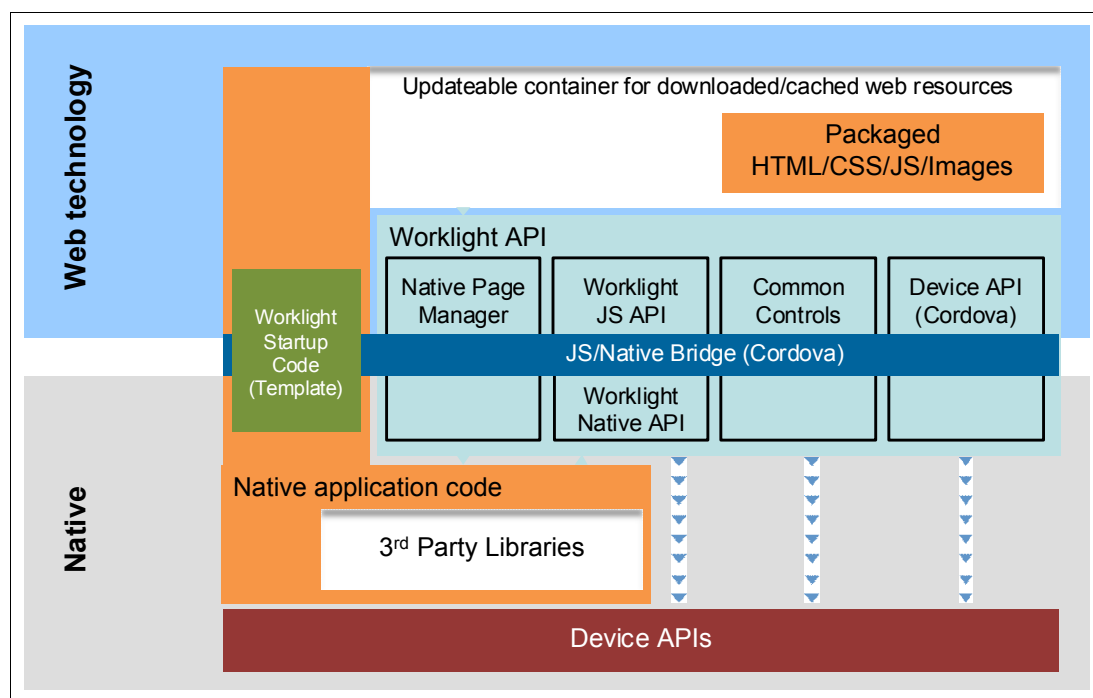


Figure 2-5 Worklight Device Runtime: Architecture overview

IBM Worklight Device Runtime supports a wide variety of mobile operating systems and release levels (some are only supported at certain fix pack levels). See the most recent list of supported mobile operating systems and release levels:

<http://bit.ly/110oTNq>

In addition to Apache Cordova, IBM Worklight Device Runtime components provide several important features that can improve application development by reducing complexity and implementation time:

- Cross-platform compatibility layer

A cross-platform compatibility layer supports development for all supported platforms. Developers of hybrid mobile applications can access common control elements, such as tab bars and clipboards, and native device capabilities such as the location service or camera. These functions can be extended for Android and iOS using a custom shell.

- Client to server integration

This feature enables transparent communication between a mobile application built using IBM Worklight technology and IBM Worklight Server. IBM Worklight forces mobile applications to use an SSL-enabled connection to the server at all times, including for authentication. This integration enables advanced management and security features such as remotely disabling a specific application version or updating the web resources of a hybrid application.

- Encrypted data store

Mobile devices are inherently insecure, so malicious application users can sometimes get access to private application data or other information. To help prevent this access, an application can store private data in an encrypted data store residing on the device and can access that data using a simple API. Using an encrypted data store prevents a malicious user from reading the information; all they get is highly encrypted data. Using ISO/IEC 18033-3 security standards such as AES256 or PKCS #5, the encryption complies with the United States National Security Agency regulations for transmitting confidential or secret information. The key that is used to encrypt the information is unique to the current user of the application and the device. IBM Worklight Server issues a special key when a new encrypted data store is created. That makes it almost impossible to reconstruct the key used to encrypt the critical information.

- JSON Store

To synchronize mobile application data with related data on the back end, IBM Worklight includes a JSON Store that provides a synchronizable, offline-capable, key-value database. It implements the application's local read, write, update, and delete operations and can use IBM Worklight adapter technology to synchronize the related back-end data. Imagine a sales representative who is making a sales call in an area without reliable network connectivity. The representative can download all customer information, sales documents, and marketing presentations before leaving the office. Then when the deal is complete and network connectivity is re-obtained, the back-end systems can be updated with all changes related to the new sale.

- Runtime skinning

This feature helps you incorporate an adaptive design that can be tailored to each mobile device. The IBM Worklight runtime skin is a user interface variant that can be applied during application run time based on device properties such as operating system, screen resolution, and form factor. This type of UI abstraction helps you develop applications for multiple mobile device models at once.

- Reporting

IBM Worklight Device Runtime has a reporting feature for application usage and user behavior. The mobile application sends events to IBM Worklight Server, which records the information in a separate database where it can be used by IBM Worklight Console or an external analytics tool such as IBM Tealeaf® CX Mobile. This allows you to study which features of your mobile application are being used and how.

- IBM Worklight APIs

These APIs provide access to IBM Worklight functions across multiple device platforms. Applications built using web technologies can access the IBM Worklight Server through these APIs using JavaScript, whereas applications that use native components can access the APIs directly using Java and Objective-C. This allows mobile applications developed with the hybrid and native development approaches, including those running Android, iOS, or Java ME, to benefit from the simplified application security and integration features of IBM Worklight.

Worklight Server

IBM Worklight Server provides a rich set of unique mobile capabilities through the use of strong client/server integration and communication between mobile applications and back-end systems. In addition, a built-in security framework allows you to use encryption and obfuscation techniques to protect both user-specific information and application business logic.

- Server-side application code

IBM Worklight Server allows you to develop server-side application code, which is valuable when performance, security, or maintenance issues make it infeasible to place all application logic on a mobile device. For instance, the mobile device may lack the computing power needed for complex calculations, but there is plenty of computing power in back end systems. With server-side application code, your mobile application has direct access to back-end transactional capabilities and cloud-based services that can improve error handling or enhance security by including additional, custom steps for request validation or process authorization.

- JSON translation

A built-in JSON translation capability transparently reduces the footprint of data transferred between the mobile application and IBM Worklight Server. JSON is a lightweight and human-readable data interchange format. Because JSON messages have a smaller footprint than other comparable data-interchange formats, such as XML, they can be more quickly parsed and generated by mobile devices. In addition, IBM Worklight Server can automatically convert hierarchical data to the JSON format to optimize delivery and consumption.

- Security framework

A built-in security framework provides easy connectivity or integration into your existing enterprise security mechanisms. This security framework handles connection credentials for back-end connectivity so the mobile application can use a back-end service without needing to know how to authenticate with it. The authentication credentials stay with IBM Worklight Server instead of residing on the mobile device. If you are running IBM Worklight Server with IBM WebSphere Application Server, you can benefit from enterprise-class security and enable single-sign-on using IBM Lightweight Third Party Authentication (LTPA).

- Adapter library

The adapter library allows you to connect to various back-end systems such as web services, databases, and messaging applications. For instance, IBM Worklight provides

adapters for SOAP or XML over HTTP, JDBC, and JMS. Additional adapters simplify integration with IBM WebSphere Cast Iron, which in turn supplies connectors for a variety of cloud-based or on-premises services and systems. With the adapter library, you can define complex lookup procedures and combine data from multiple back-end services. This aggregation helps to reduce overall traffic between a mobile application and IBM Worklight Server.

- **Direct Update**

Another unique feature of IBM Worklight Server is Direct Update, which enables your web or hybrid application to update its web resources to the latest versions. This feature helps you manage application versions from your mobile middleware. Whenever the mobile application connects to IBM Worklight Server, the server can deny or allow access or force the user to update to a new version. This can be done with or without the user's knowledge or consent; silent update rollouts can be beneficial in that they avoid interrupting the users and reduces the risk that they might lose unsaved information when initiating the update.

- **Unified push notification**

Unified push notification is an abstraction layer for sending notifications to mobile devices using either the device vendor's infrastructure or IBM Worklight Server's SMS capabilities, which simplifies the notification process because the application can remain platform-neutral. The user of a mobile application can subscribe to notifications through the mobile application. This request, which contains information about the device and platform, is then sent to IBM Worklight Server. The system administrator can manage subscriptions, push or poll notifications from back-end systems, and send notifications to mobile devices using Application Center.

- **Reporting**

IBM Worklight Server provides reporting capabilities to help you better understand your application's installation base with detailed information about application activity and the mobile environments where your application is running. IBM Worklight Server automatically collects information about the user and device (including manufacturer and unique device identifier), the application and version, and IBM Worklight adapters that are being called. Additional log activities can be added to an application to collect extra information. IBM Worklight Server includes several basic reports that can be used with the Business Intelligence Reporting Tool (BIRT), which is available from Eclipse. And because the collected information is stored in a separate database, it can be accessed by other analytics tools, such as IBM Cognos® Business Intelligence, for deeper analysis.

Worklight Console

The IBM Worklight Console is a *web-based administrative interface* for IBM Worklight Server. It displays all deployed applications, adapters, and push notification rules.

Features are as follows:

- **Management of mobile application releases**

You can use the application management functionality of IBM Worklight Console to manage mobile application releases by deactivating outdated application versions based on pre-established rules. You can send messages to users to inform them when a new release is available or when there are fixes for known security issues. In addition, you can assign device-specific identifiers to ensure secure application provisioning and device identification, including improved storage encryption.

- **Visual interface**

IBM Worklight Console simplifies management of adapters and provides a visual interface for deployment and update functions for adapters (Figure 2-6 on page 39). This view lists

all available adapters and, for each one of them, shows all of the operations that a mobile application can invoke.

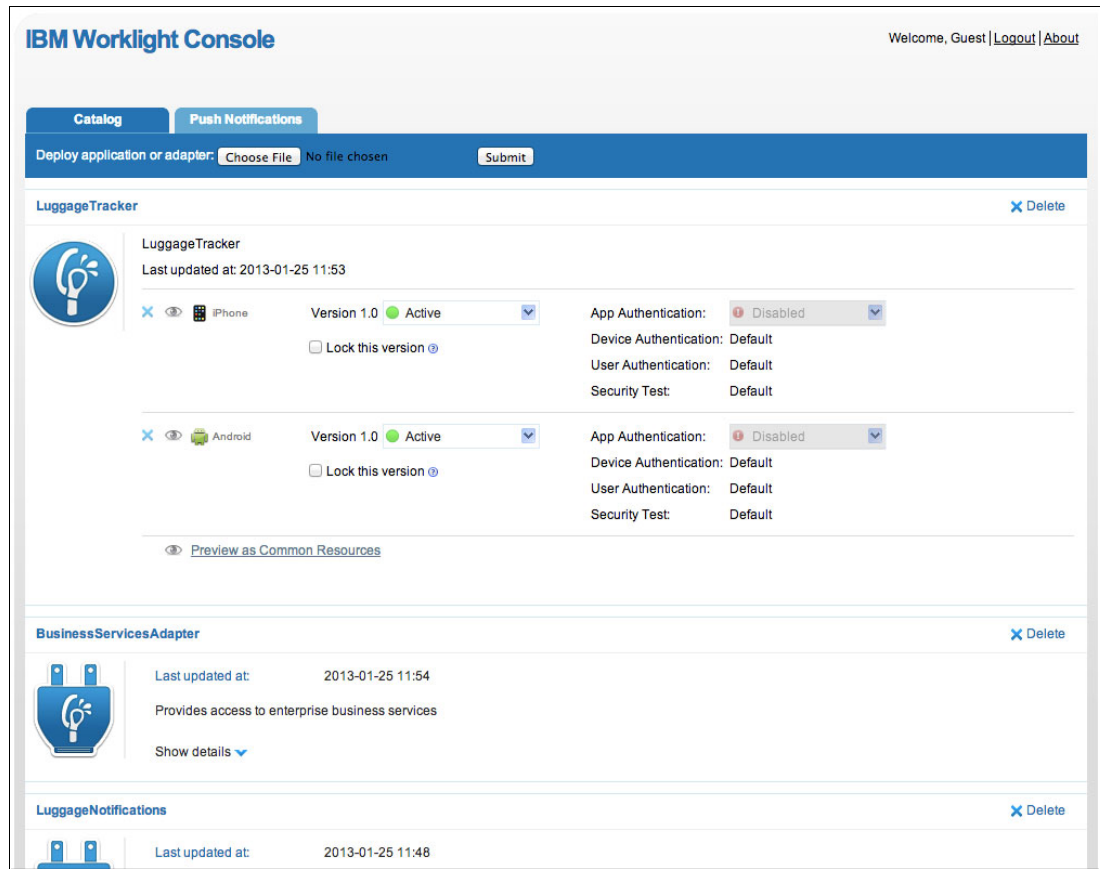


Figure 2-6 IBM Worklight Console showing the catalog of applications and adapters

► View statistical user information

The console can also be used to view statistical user information collected from all running applications by IBM Worklight Server. User adoption and usage reports can help your organization make decisions regarding the value of specific platforms and support for specific user groups.

Worklight Application Center

IBM Worklight Application Center is a web-based, internal enterprise application store for centralized application distribution, installation, and feedback.

The architecture of Worklight Application Center is shown in Figure 2-7 on page 40 and demonstrates the central role of an application store. The application center offers three main services:

- An application catalog service to search for available mobile applications
- An application installation service to install mobile applications
- An application feedback service to provide feedback on application versions

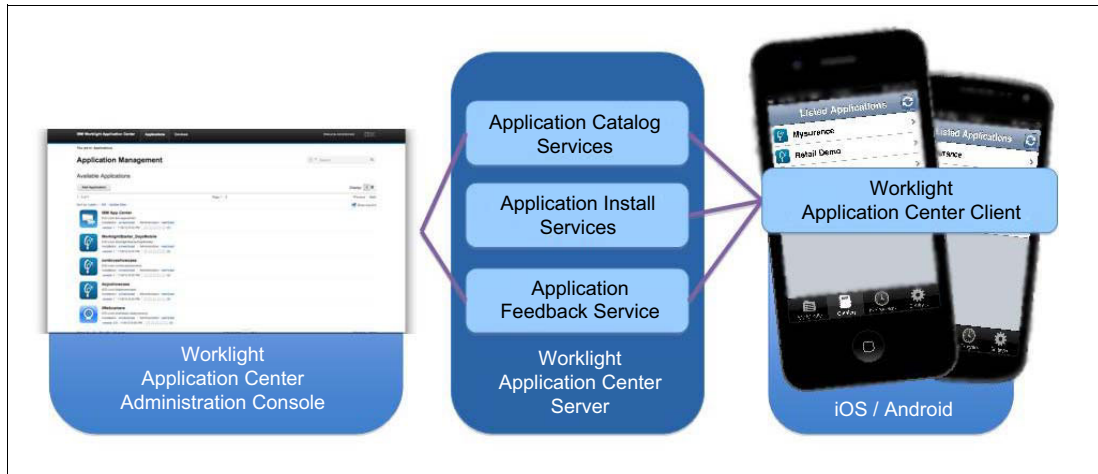


Figure 2-7 Worklight Application Center architecture

With an internal enterprise application store (using the application catalog service and application install service), internal users can view your catalog, download applications, track installed applications, and provide feedback by rating the application versions. Allowing an employee to manage the applications on a company-owned mobile device instead of requiring direct IT support can reduce IT support costs.

The application feedback service enables users to rate an application and provide feedback about it. This helps you monitor user acceptance and plan for new releases. For example, if one feature receives constant positive feedback, you can shift development resources to improve other features that are not as highly rated or get negative feedback.

During the development lifecycle, Application Center can be used to streamline the movement of new application versions from development to test, such as when you must release a new version to an internal test audience. Granular access controls allow you to offer multiple versions of an application in the internal application store but to limit access to specific versions to certain users or groups.

2.3.2 WebSphere CastIron Hypervisor Edition

IBM WebSphere Cast Iron Hypervisor Edition is a virtual appliance for service integration. It provides connectivity to a variety of cloud-based and on-premises applications, databases, web services, messaging systems, and other endpoints. In addition to predefined endpoint connectors, IBM WebSphere Cast Iron Hypervisor Edition includes templates that enable access to other endpoints such as Google Analytics, Microsoft Azure Servicebus, or Oracle CRM on Demand.

IBM WebSphere Cast Iron Hypervisor Edition uses a configuration approach in which integration projects are configured instead of programmed. With a configuration approach, mobile applications can be connected to back-end systems without any help from a developer. Instead, you build the needed integration flows using a graphical development environment in WebSphere Cast Iron Studio.

Each integration project consists of one or more *orchestrations*, each of which describes a specific set of activities that define the flow of data, including access to back-end systems and data transformations. Mobile applications can use an IBM Worklight Cast Iron adapter to perform the activities defined in an orchestration.

Transforming data from its original format to another format can be done using a simple drag-and-drop method in the WebSphere Cast Iron Studio graphical mapping editor, which is shown in Figure 2-8. There is no need to understand all supported data formats because the data transformations are done visually using the mapping editor.

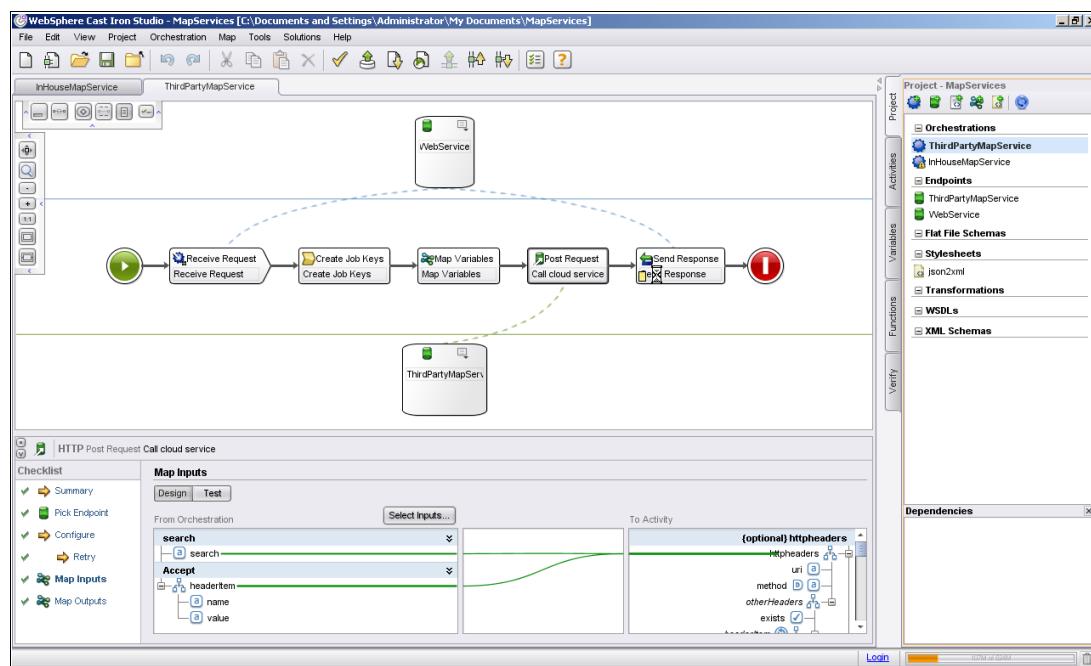


Figure 2-8 Using the IBM WebSphere Cast Iron Studio graphical mapping editor

Business logic rules also can be set up using the graphical mapping editor. Because it is a WYSIWYG (what you see is what you get) interface, business users can participate in an integration project without any specific IT skills.

For administration, WebSphere Cast Iron Hypervisor Edition provides a web-based console to monitor the data flow, handle exceptions in the data flow, and provide proactive alerts regarding data and connectivity errors. The console also provides easy access to key performance indicators.

IBM WebSphere Cast Iron provides a secure environment with capabilities to easily and rapidly assemble and publish useful business services. Re-using these existing business services (and their existing infrastructure) can reduce a new application's time-to-market. A company can also offer these business services to other companies to open new market opportunities.

More information about IBM WebSphere Cast Iron is at the following address:

<http://www.ibm.com/software/integration/cast-iron-cloud-integration/features>

2.3.3 IBM WebSphere DataPower appliances

IBM WebSphere DataPower appliances are designed to support the implementation of enterprise solutions by introducing security layers, providing application integration capabilities, and enhancing overall integration performance.

The main advantage of WebSphere DataPower appliances is that they are easy to integrate into a network infrastructure, where they provide a software-independent configuration and

simplified functionality. In addition, in development environments, some appliance virtual machines can be used to accomplish the same basic functions of the real appliances.

When used as the security layer provider, IBM WebSphere DataPower appliances can execute many critical security tasks by off-loading them from the application server.

Key security-related capabilities of IBM WebSphere DataPower appliances are as follows:

- ▶ Unloading HTTPS session handling from the web server
- ▶ Acting as web application and XML firewall
- ▶ Providing authorization, authentication, and auditing (known as *AAA*) in a single mechanism
- ▶ Implementing an enterprise single sign-on (SSO) function through the use of LTPA tokens

In terms of application integration and performance enhancement, IBM WebSphere DataPower appliances provide the following functions:

- ▶ Unloading XML, XSLT and XPATH processing from the application server to the DataPower appliance and performing data transformations with better response times
- ▶ Acting as an enterprise service bus (ESB) to provide integration between different architecture layers
- ▶ Accelerating data conversion and application integration
- ▶ Providing support for transmission of specialized business-to-business message traffic between partners

For a complete list of the currently available IBM WebSphere DataPower appliances and their functions, view the product details at this address:

<http://www.ibm.com/software/integration/datapower>

2.3.4 IBM WebSphere eXtreme Scale and the IBM WebSphere DataPower XC10 caching appliance

The IBM WebSphere caching family includes IBM WebSphere eXtreme Scale and the IBM WebSphere DataPower XC10 caching appliance. The main difference between the two solutions is that IBM WebSphere eXtreme Scale is purely a software-based solution, and IBM WebSphere DataPower XC10 is a mixed firmware solution delivered as a caching appliance.

IBM WebSphere eXtreme Scale provides a flexible way to cache both business logic and application data between multiple servers in a fault-tolerant, scalable solution. The key benefits of using this product are as follows:

- ▶ Reduced latency and process overhead compared to a traditional caching solution
- ▶ Improved response time through the elimination of disk operations
- ▶ Horizontal and vertical scaling to meet requirements for high performance caching, including deployment in cloud environments to add power to the caching solution

In contrast, the IBM WebSphere DataPower XC10 caching appliance provides a ready-to-use, plug-in caching solution that can rapidly fulfill an application's caching needs without affecting the programming model. A high availability caching system can be implemented by associating multiple devices to work as a logical group.

The key benefits of the DataPower XC10 caching appliance are as follows:

- ▶ Ease of implementation, as there is no need to make changes in the application and no programming is involved, allowing for fast deployment times
- ▶ Variable device topologies, which can be implemented to achieve the required scalability levels
- ▶ Simplified management and security

Although both WebSphere eXtreme Scale and the WebSphere DataPower XC10 caching appliance are designed to implement the caching tier of an enterprise application, certain scenarios are more suitable to one particular product:

- ▶ The IBM WebSphere DataPower XC10 caching appliance is preferred when the aim is to implement a simple, data-oriented caching solution that is fast to deploy. Ideal fits for this solution include HTTP session management, extending DynaCache, or as a simple ESB cache.
- ▶ IBM WebSphere eXtreme Scale can be applied to the same scenarios as the caching appliance but offers additional possibilities such as providing a caching solution for database buffering, business event processing, and service-oriented architecture (SOA) operations that require in-memory response times.

The IBM WebSphere DataPower XC10 caching appliance is the best choice for a standard caching implementation although IBM eXtreme Scale is ideal for more sophisticated, multiple-mechanism caching solutions.

For more information about IBM WebSphere eXtreme Scale, visit the product website:

<http://www.ibm.com/software/webservers/appserv/extremescale/>

For details about the IBM WebSphere DataPower XC10 Appliance, visit the product website:

<http://www.ibm.com/software/webservers/appserv/xc10/>

2.4 Adapting your mobile solution to a different audience

Your mobile solution can have various potential audiences (see 1.2.1, “Business-to-consumer” on page 5 and 1.2.2, “Business-to-employee” on page 6). The appropriate configuration for you depends on your intended audience. The two configurations used in this book are as follows:

- ▶ Mobile enterprise application platform
- ▶ Mobile consumer application platform

2.4.1 Mobile enterprise application platform configuration

IBM Worklight can be used as a mobile enterprise application platform to allow employees to use mobile devices to access enterprise resources, such as collaboration tools or business information systems. IBM Worklight Application Center is used as an internal application store to distribute mobile applications inside your organization, although IBM Endpoint Management for Mobile Devices addresses the endpoint management challenges inherent in mobile device management.

Figure 2-9 shows a typical architecture for a business-to-employee scenario.

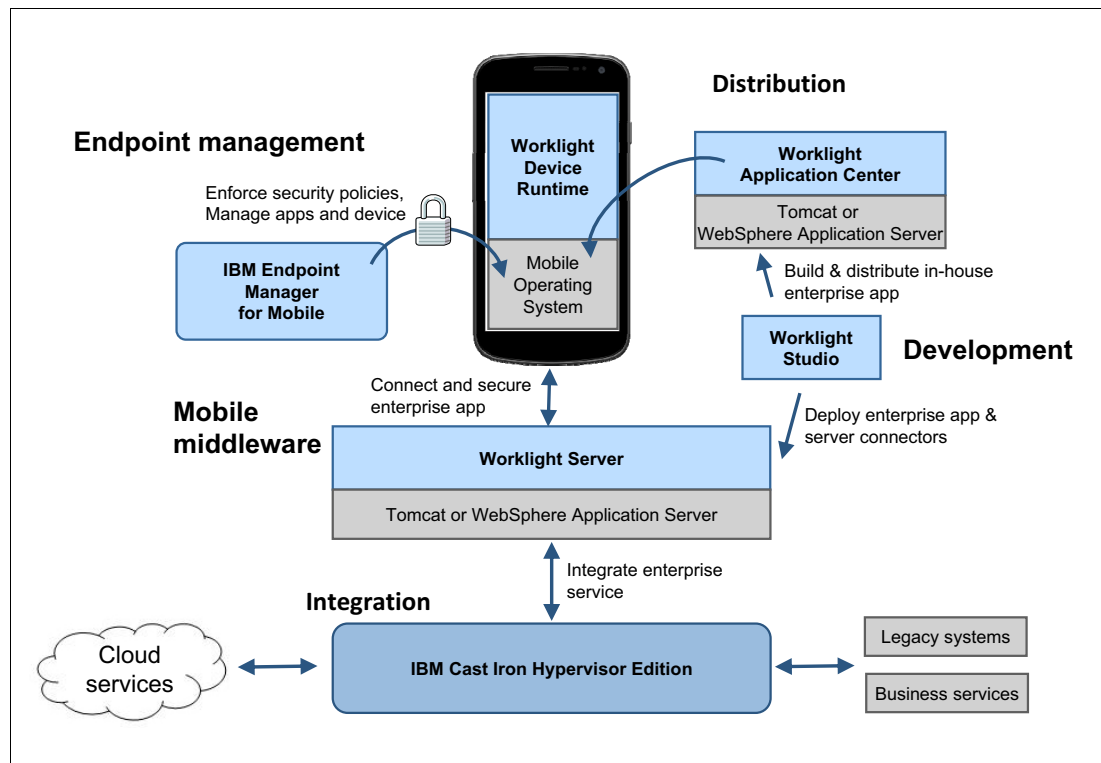


Figure 2-9 Typical business-to-employee configuration

In many cases, your infrastructure already handles a well-defined user base (your employees). So starting your mobile solution with a business-to-employee scenario allows you to work with your existing user base and then scale up with additional Worklight servers as you expand into business-to-consumer scenarios, where your user base will be more volatile.

With Worklight Studio, you can push developed mobile applications straight to your IBM Worklight Server and IBM Worklight Application Center for fast distribution. IBM Worklight Application Center uses an access-control list (ACL) to control which users can install a specific mobile application.

Using WebSphere Cast Iron Hypervisor Edition enables your mobile application platform infrastructure to easily integrate with pre-existing systems or utilize cloud service offerings. It helps you integrate with different back-end systems, such as Siebel or SAP, and talks with a variety of protocols, such as SOAP/HTTP, JMS, and FTP. WebSphere Cast Iron Hypervisor Edition offers service-oriented architecture to integrate various services within your organization.

If your IT systems communicate with external systems, such as for push notifications or connections to cloud services, a firewall is used to control the inbound and outbound connections. For these connections, consider protecting your internal IT infrastructure with a proxy or security gateway such as IBM WebSphere DataPower XML Security Gateway XS40. Because external requests, such as denial of service attacks, can compromise your infrastructure, they can affect more than just your internal business processes, they can have financial impacts, too.

2.4.2 Mobile consumer application platform configuration

For scenarios involving consumers, IBM Worklight and IBM WebSphere CastIron work together to provide a streamlined and tailored solution. Because this topology targets consumers, the distribution of mobile applications is performed outside of your organization. You instead have to deal with the submission process for the Apple, Google, and RIM application stores, among others.

Figure 2-10 shows a typical business-to-consumer topology.

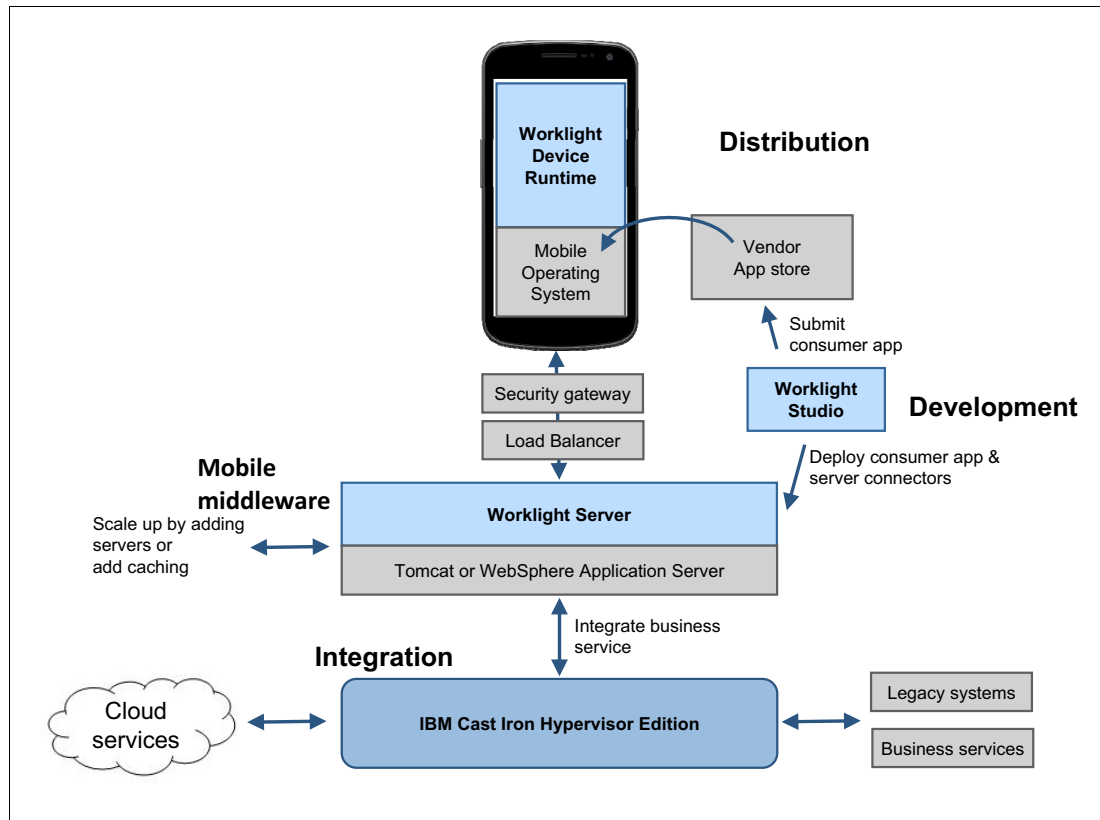


Figure 2-10 Typical business-to-consumer configuration

The biggest differences between this configuration and the business-to-employee one depicted in Figure 2-9 on page 44 lie in the areas of quality of service, scalability, and infrastructure protection. For instance, your mobile application will be available to potentially hundreds of millions of customers as soon as it is listed in a vendor's app store. Even if only a fraction of those customers use your offering, millions of new requests can come into your infrastructure. In response, you can either increase your available service capacities to handle the new peak load or you can reduce quality of service factors such as response times and queue requests. Be careful what you choose: a good reputation in the vendor's app stores attracts communities of users and media interest, and all the other products of your organization will be moved into this spotlight, for good or bad.

It is important to note that these components are scalable and run in virtual environments. If your application generates positive consumer feedback, your user base can dramatically increase within just a few days and you must adapt accordingly. To optimize server utilization, your infrastructure must be *elastic* regarding consumer requests. This means that your infrastructure must be able to add more resources to handle increased requests, but also

remove these additional resources as requests decrease. Virtualization addresses the issue of server utilization and must be considered when you plan your infrastructure.

Using a security gateway, such as IBM WebSphere DataPower XML Security Gateway XS40, enables you to control the parts of your infrastructure that are exposed to consumers. Using a perimeter network (sometimes referred to as a DMZ), which exposes only customer-facing services to the Internet, works well for inbound connections. But to use push notifications, your server must create inbound and outbound connections to and from IBM Worklight Server. Consequently, if your security gateway is compromised, a hacker can move in to attack your internal resources.



Expanding the enterprise using mobile technology

This chapter illustrates a real-world scenario using a fictional enterprise named Fictitious Airline Company A. This chapter introduces the company and takes you through the process that the company follows as it decides to expand into the mobile space.

The following topics are covered:

- ▶ 3.1, “Introducing Airline Company A” on page 48
- ▶ 3.2, “IT department roles” on page 48
- ▶ 3.3, “Business goals and performance indicators” on page 50
- ▶ 3.4, “Building the mobile roadmap” on page 51
- ▶ 3.5, “Building the mobile strategy” on page 53
- ▶ 3.6, “Selecting a Mobile Application Platform” on page 54

3.1 Introducing Airline Company A

Fictitious Airline Company A is the flagship and national carrier of Country A. It was founded 50 years ago and now operates on five continents, with numerous daily flights between domestic and international destinations. Airline Company A became one of the world's leading airlines by generating unprecedented growth in profit over the past five decades. Airline Company A was an integral part of boosting the tourism industry in its region tenfold.

The company employs approximately 25,000 employees around the world with staff in most major airports, and serves an estimated 50 million passengers annually.

Airline Company A prides itself on operational efficiency and outstanding customer service, and it has consistently generated high profits by offering low-cost, high-quality services to passengers. But, as with all airlines, the company must continuously improve profitability by gaining economic efficiencies. And despite being a traditional organization, the competitive passenger transportation industry is forcing the airline to seek new ways to differentiate itself from the competition. At the same time, it must handle increasing fuel prices and maintain the kind of quality customer service that helps ensure high passenger demand.

3.2 IT department roles

The primary focus of the IT department of Airline Company A is to ensure efficient technology processes and to ensure that the business operational model is efficiently managed. The IT organization is made up of individuals who specialize in certain areas or roles.

3.2.1 Previous roles

Airline Company A has always been a traditional organization in terms of its organizational structure. Figure 3-1 shows the structure of the company's IT department before it started thinking about entering the mobile space.

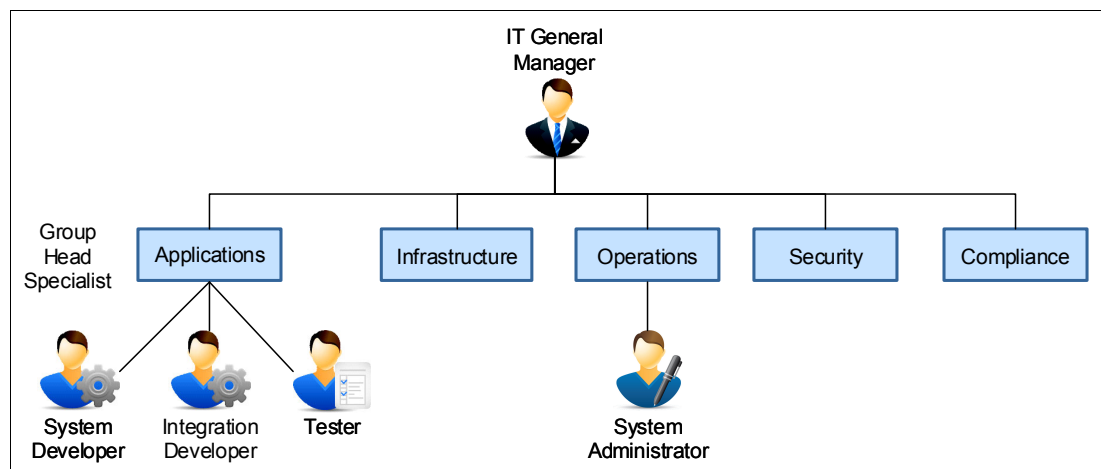


Figure 3-1 Previous IT organizational structure

The previous IT organization was lead by a general manager who oversaw the following divisions, each with its own organizational sub-structure:

- ▶ Applications, including various system and integration developers and application testers
- ▶ Infrastructure
- ▶ Operations, including the system administrator
- ▶ Security
- ▶ Compliance

3.2.2 New roles to support mobile technologies

For Airline Company A to begin offering mobile applications and services, the company reconfigured its IT organization to ensure it would have the correct people and skill sets in place. The new IT structure is outlined in Figure 3-2.

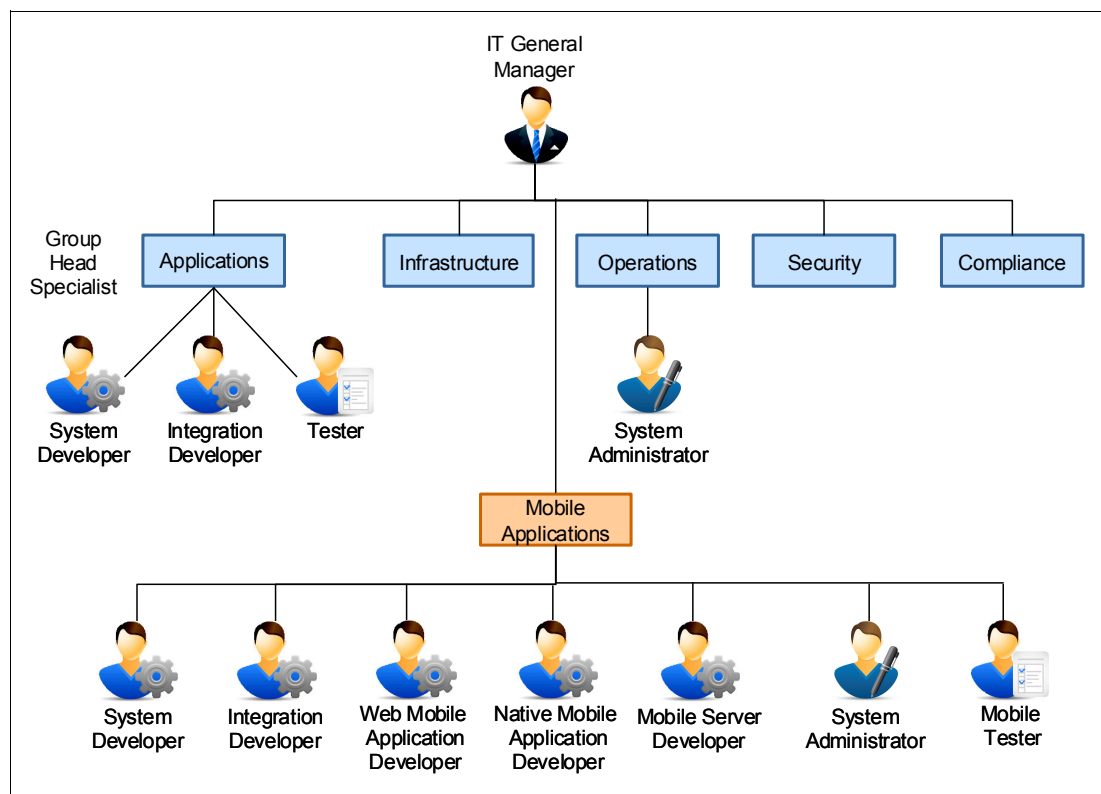


Figure 3-2 New IT organizational structure

The main difference in the new IT department of Airline Company A is the addition of a Mobile Applications division. And although the roles within the new division are traditional IT-related roles, they are specifically targeted to mobile application development and maintenance. For example, to implement its first mobile application, Airline Company A needed mobile application developers and mobile server developers.

The specific characteristics and areas of expertise and responsibility for each new mobile-related IT roles are listed here:

- ▶ A *system developer* is responsible for the implementation of business applications. These business applications use Java and Java Enterprise Edition technologies, such as web services and Enterprise Java Beans (EJB).

There are system developers with similar skills in both the Applications and Mobile Applications divisions. This role could have been limited to the Applications division, but the company opted to make its new Mobile Applications division self-contained. A system developer in the Mobile Applications division works on mobile applications only.

- ▶ An *integration developer* is knowledgeable in a variety of protocols and formats and is responsible for an integration layer, that makes existing business applications and functions (provided by the system developer) available as services, in a secure manner. The skill set of an integration developer in the Applications division is similar, but the integration developer for the Mobile division must take into account the small footprint requirements and unique formats for mobile devices.
- ▶ A *web mobile application developer* uses special knowledge of web technology to create an application that can be installed and run on mobile devices. The mobile application developer works with the mobile system developer to ensure that the mobile application can utilize the adapter layer provided by the mobile server developer.
- ▶ A *native mobile application developer* uses special knowledge of a specific vendor's mobile environments to create or enhance an application with features that run only on that vendor's platform. Like the web mobile application developer, the native mobile application developer also collaborates with the mobile system developer.
- ▶ A *mobile server developer* works in collaboration with the mobile application developer and the integration developer to create an adapter layer that links the mobile application to the available services. An adapter is the server-side function that connects a mobile application to the web services in the integration layer, and performs any necessary server-side logic on the data flowing to and from the mobile application. The mobile server developer is also responsible for the implementation of security mechanisms that are unique to mobile devices and integrating those mechanisms into the existing security infrastructure. The mobile server developer also incorporates unique mobile technologies such as push notifications, into the adapter layer.
- ▶ A *system administrator* maintains all of the systems within the production environment that are unique to the mobile solution, including the mobile application platform, the adapter layer and the integration layer. This role is also responsible for application management, including the deployment of the mobile application to an application store, quality of service monitoring, and reporting.
- ▶ The *mobile tester* role is similar to the tester role in the Applications division, but the mobile tester has a unique skill set to test on both device simulators and actual mobile devices utilizing all of the supported device platforms.

3.3 Business goals and performance indicators

The vision of Airline Company A is to be the leading aviation company in its region. The company plans to achieve this by offering safe and reliable services with superior customer experience. Several goals of Airline Company A are as follows:

- ▶ Ensuring that all customers receive a high-quality flying experience
- ▶ Improving customer retention by targeting the individual needs of passengers

- ▶ Increasing the customer base and expanding the number of destinations by attracting new flyers, particularly young flyers, who have not previously flown with Airline Company A
- ▶ Reducing engine emissions
- ▶ Decreasing the time passengers spending waiting at an airport
- ▶ Improving efficiency by automating important repetitive tasks
- ▶ Increasing the productivity of staff within the airline and its business ecosystem
- ▶ Growing profits by decreasing operational expenses and increasing revenue

Airline Company A believes that the key differentiator between it and the competition is not only ticket prices but the end-to-end passenger experience. So it is undertaking a business strategy that is focused on implementing an automated mobile solution.

3.4 Building the mobile roadmap

Airline Company A knows that a mobile solution requires a comprehensive and holistic strategy, so it is building its mobile strategy using the domain model as described in 1.3, “Mobile challenges” on page 7.

The first two domains of the domain model, discussed here, help the company build its overall mobile roadmap, including both short-term and long-term plans. The final four domains, discussed in 3.5, “Building the mobile strategy” on page 53, help the company expand its roadmap into a full-fledged mobile strategy.

3.4.1 Mobile transformation (building the roadmap)

The objective for implementing a mobile solution for the company is to extend many of its existing business capabilities to mobile devices. These capabilities include the following items:

- ▶ Luggage tracking
- ▶ Flight reservations and cancellations
- ▶ Social media interactions
- ▶ Visa application assistance
- ▶ Offering information from elsewhere in the airline’s business partner ecosystem such as hotel address and amenities

All of these capabilities are independent features and functions that Airline Company A currently provides to its customers. This functional independence allowed the company to start by extending only one capability to the mobile space. Then, based on the success of the initial effort, additional capabilities will be extended to mobile devices.

Therefore, for the first phase of its mobile roadmap, Airline Company A decided to launch a single, business-to-employee application for luggage tracking. This application would enable customer service representatives to view the current location of a passenger’s luggage at any time. Luggage tracking was considered an ideal choice for an initial solution because it does not require any process changes that might disrupt current customers and is not expected to be too demanding on the IT department.

In addition, concentrating on an employee-focused application allowed the company to use the project as a training ground for possible future mobile-related efforts.

This stage of the process also requires an enterprise to define the success criteria for any new mobile solution. Airline Company A decided to measure its new luggage tracking solution according to these key performance indicators:

- ▶ Increase in staff productivity
- ▶ Decrease in passenger complaints regarding luggage whereabouts
- ▶ Staff feedback about the mobile solution

Airline Company A's existing distributed infrastructure environment adheres to a service-oriented architecture (SOA) concept, which includes well-defined services. This allowed the company to adopt mobile technology more easily because these well-defined services provide access to much of the needed information. Because many mobile application users expect rapid delivery of newer versions (with updated features), Airline Company A chose to adopt the SCRUM agile development lifecycle (see Figure 2-1 on page 23) for developing its new mobile application. A small number of features will be included in each sprint, and will be released at the end of that sprint, thereby reducing the delivery time of features to the application users.

3.4.2 Mobile scenarios and user groups (who will use the app and how)

The first application will be for luggage tracking. Arriving passengers who are trying to locate their checked luggage will be able to seek assistance from a customer service representative who can then determine the location of the customer's bags using the new application installed on his or her mobile device.

As input, the mobile application depends on a unique luggage identification code that was provided to the passenger at check-in. So the customer service representative scans the identification code (or enters it manually) and then submits a tracking request using the application. The application then connects to the airline's existing back-end baggage tracking systems to determine the location of the customer's specific piece of luggage. If the bag has not arrived yet, the representative can input a delivery address so that the bag can be delivered to the passenger's home or hotel upon later arrival.

As part of its comprehensive mobile roadmap, Airline Company A plans additional phases to cover additional scenarios, which will be implemented after the first phase is completed and the company has obtained feedback on the luggage tracking application. These extra scenarios add the following capabilities:

- ▶ Enable customers to track their luggage themselves; if the luggage is delayed, the customers can add and update a delivery address. This effectively takes the first-phase functionality, which was available only to customer service representatives, and puts it in the hands of the customer.
- ▶ Integrate with social media so passengers can compile a list of their interests and make the list available to other passengers. This information can be valuable to customers during seat selection; a passenger may choose a seat assignment based on the interests of neighboring passengers.
- ▶ Enable notifications through the mobile application so that, for example, the airline can proactively notify passengers in the event of a flight delay. Such a notification could be sent as soon as a delay is known so passengers could choose to delay their arrival at the airport based on the new departure time.

3.5 Building the mobile strategy

After the mobile roadmap is devised, a more detailed mobile strategy is built. Airline Company A has chosen to develop its mobile strategy use of the final four domains in the domain model.

3.5.1 Mobile device selection

To reduce its initial investment, Airline Company A will use the bring your own device (BYOD) policy and have its customer service representatives run the new mobile application on their personally owned mobile devices.

Most employees own Apple iOS or Google Android mobile devices, so the initial application will support these two platforms only. In addition, the luggage tracking application will depend on using the native camera and geo-data applications present on most mobile devices.

Therefore, the minimum requirements for the employee-owned devices is that they run either an Apple iOS or Google Android operating system and have a camera and geo-data-based location services.

3.5.2 Mobile applications

The initial luggage tracking application will be developed using a hybrid development approach. The following factors contributed to this decision:

- ▶ The company can create the mobile application using existing skills in web application development, reducing development time and costs.
- ▶ The luggage tracking application can run on the required platforms although platform specific capabilities, such as the camera or location services, can still be accessed without the cost of maintaining two separate native applications.
- ▶ The web portions of the mobile application can be updated using the Direct Update feature of Worklight, without requiring users to reinstall a new mobile application.

3.5.3 Mobile integration

Airline Company A investigated its existing IT infrastructure and determined that some of the data that will be required by the mobile luggage tracking application, such as luggage identification and passenger information, is already available in existing systems. So a new integration layer will be created to access this existing data and impose strict guidelines on its use. The integration layer will be responsible for these tasks:

- ▶ Retrieving information from, and providing data to, existing systems.
- ▶ Reducing the amount of information to be transferred between the mobile application and existing systems by omitting data fields that are not required by the application on the mobile device. For example, the application does not need to know how the customer paid for his or her airline ticket.
- ▶ Transforming data from the format in the existing systems to whatever format is needed by the mobile application.

The mobile application will also use the integration layer to access enterprise services, because the company's risk management and security guidelines prohibit direct communication between a mobile application and the company's back-end systems.

The integration layer consists of the server-side functions of the mobile application platform. These functions can include additional business logic that is needed only by the new mobile application and the adapters that are required to transform the data.

The integration layer can be used to enhance the scalability of the existing infrastructure by creating caching mechanisms within the integration layer. These caching mechanisms can be used to provide scalable components that can address the changing needs of mobile solutions and therefore, handle any volatile usage. This is not needed for the initial luggage tracking application, but the company included caching in its roadmap requirements for its business-to-consumer scenarios.

3.5.4 Mobile operations

The IT department of Airline Company A defined operational guidelines as part of the mobile strategy. The company decided that mobile applications will access back-end services through the integration layer, using the existing security infrastructure. Authentication will be verified against the company's directory service by using a WebSphere DataPower XG45 appliance. No additional mobile-specific security mechanisms will be required to start.

The mobile application platform will assist the company's mobile team in regulating the release and update cycle of mobile applications. For business-to-employee applications, the mobile application platform is used to deliver and update the application using the built-in Application Store feature. For the business-to-consumer application, the application will be released through a vendor application store but the web (common) portions of the application can be updated using the mobile application platform.

Support for users of the mobile application will be provided with a built-in feedback mechanism. Users can use the feedback mechanism to submit written questions that will be routed to the existing IT support team. No specific call center support for this application will be provided. This approach is within the accepted leading practices for mobile solutions.

3.6 Selecting a Mobile Application Platform

When its mobile strategy is defined, Airline Company A is ready to select a Mobile Application Platform (MAP). Although the initial luggage tracking application will be used by employees only, their roadmap includes expanding into the business-to-consumer application space also. To ensure that the application platform can handle both internal and external applications, Airline Company A decided to select a Mixed Mobile Application Platform (MMAP). The company selected IBM Worklight as its base for the MMAP, primarily because Worklight features and capabilities support development, integration, and operations.

From a development perspective, IBM Worklight allows fast and easy development of hybrid applications using IBM Worklight Studio and the applications can be run on multiple platforms using the IBM Worklight Device Runtime component. Using IBM Worklight technology, mobile application developers can enrich mobile applications with native components such as platform-specific UI widgets. The mobile application developers can benefit from the graphical, rich text editor in IBM Worklight Studio to easily develop the mobile application's user interface and can integrate third-party libraries such as jQuery Mobile and Dojo Mobile. In addition, the adapter editor in IBM Worklight Studio allows the development team to easily create the integration layer of the MAP. And, the integrated test server and mobile device simulator in Worklight allow early testing of the application without requiring each developer and tester to have a mobile device in their hands.

From the integration perspective, IBM Worklight adapters can provide easy integration of business services and can be used to connect the mobile application to existing enterprise services. Security features, such as Lightweight Third Party Authentication (LTPA) and custom security mechanisms allow flexibility to integrate mobile applications with a company's existing security infrastructure. IBM Worklight supports the required lightweight data interchange formats, such as JSON, to reduce the size of the data being transmitted from the mobile application to the server. Furthermore, the unified cross-platform notification feature allows an enterprise to handle the complexity of contacting all vendor-specific notification services outside their enterprise business logic. With this mechanism, companies can send notifications to all devices that have the mobile application installed. In addition, the notification feature can be configured to send the notification regardless of the operational state of the mobile application.

From an operational perspective, IBM Worklight provides a company with an internal application store to distribute mobile applications to the employees. It allows a company to centrally manage its mobile applications by rolling out updates and remotely disabling outdated versions. In addition, the integrated Application Center of IBM Worklight allows for collecting feedback from users. The reporting and analytics capabilities of IBM Worklight allow a company to determine which functions within an application are most used, which can help the company make decisions regarding whether to increase resources to match rising application usage. IBM Worklight can also be integrated with data caching products to allow temporary storage of data in the MMAP.



Part 2

Scenario introduction and technical implementation

This part describes the business scenario that is used in this IBM Redbooks publication. The book uses a fictitious company and describes the company's business goals and vision to expand into the mobile space. The book leads you through the thought process in designing the mobile application, and also the impact to an existing enterprise topology.

Also in this part are the step-by-step details of the implementation, and the entire lifecycle, from design and development through deployment and maintenance.

This part contains the following chapters:

- ▶ Chapter 4, "Creating and deploying the mobile solution" on page 59
- ▶ Chapter 5, "Expanding the solution to consumers" on page 205
- ▶ Chapter 6, "Installation and configuration" on page 293



Creating and deploying the mobile solution

This chapter details the implementation of the mobile luggage tracking solution for Airline Company A that is introduced in 3.4.1, “Mobile transformation (building the roadmap)” on page 51. The chapter assumes that you have basic knowledge of general application development but have not previously developed mobile applications. It also assumes knowledge of Eclipse and its functions. The iOS-specific sections assume that you have knowledge of the Xcode IDE.

The following sections guide you through the design, creation, and deployment of the new mobile solution:

- ▶ 4.1, “Breaking down the scenario” on page 61
- ▶ 4.2, “Planning and setting up the environments” on page 67
- ▶ 4.3, “Providing enterprise services” on page 71
- ▶ 4.4, “Planning for security” on page 75
- ▶ 4.5, “Operating the Mobile Enterprise Application Platform” on page 78
- ▶ 4.6, “Introducing Worklight Studio projects” on page 87
- ▶ 4.7, “Creating the adapters” on page 88
- ▶ 4.8, “Creating the mobile application” on page 104
- ▶ 4.9, “Deploying the application” on page 170
- ▶ 4.10, “Testing the end-to-end scenario” on page 179
- ▶ 4.11, “Using feedback mechanisms” on page 189
- ▶ 4.12, “Analyzing application and adapter usage” on page 191

This chapter includes snippets of source code used in the mobile solution.

IBM DOES NOT WARRANT OR REPRESENT THAT THE CODE PROVIDED IS COMPLETE OR UP-TO-DATE. IBM DOES NOT WARRANT, REPRESENT OR IMPLY RELIABILITY, SERVICEABILITY OR FUNCTION OF THE CODE. IBM IS UNDER NO OBLIGATION TO UPDATE CONTENT NOR PROVIDE FURTHER SUPPORT.

ALL CODE IS PROVIDED "AS IS," WITH NO WARRANTIES OR GUARANTEES WHATSOEVER. IBM EXPRESSLY DISCLAIMS TO THE FULLEST EXTENT PERMITTED BY LAW ALL EXPRESS, IMPLIED, STATUTORY AND OTHER WARRANTIES, GUARANTEES, OR REPRESENTATIONS, INCLUDING, WITHOUT

LIMITATION, THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY AND INTELLECTUAL PROPERTY RIGHTS. YOU UNDERSTAND AND AGREE THAT YOU USE THESE MATERIALS, INFORMATION, PRODUCTS, SOFTWARE, PROGRAMS, AND SERVICES, AT YOUR OWN DISCRETION AND RISK AND THAT YOU WILL BE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY RESULT, INCLUDING LOSS OF DATA OR DAMAGE TO YOUR COMPUTER SYSTEM.

IN NO EVENT WILL IBM BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY OR CONSEQUENTIAL DAMAGES OF ANY TYPE WHATSOEVER RELATED TO OR ARISING FROM USE OF THE CODE FOUND HEREIN, WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, LOST SAVINGS, LOSS OF PROGRAMS OR OTHER DATA, EVEN IF IBM IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS EXCLUSION AND WAIVER OF LIABILITY APPLIES TO ALL CAUSES OF ACTION, WHETHER BASED ON CONTRACT, WARRANTY, TORT OR ANY OTHER LEGAL THEORIES.

THIRD PARTY SOFTWARE IS LICENSED AND DISTRIBUTED TO YOU BY THE THIRD PARTY DISTRIBUTORS AND/OR RESPECTIVE COPYRIGHT AND OTHER RIGHT HOLDERS UNDER THEIR TERMS AND CONDITIONS. IBM MAKES NO EXPRESS OR IMPLIED WARRANTIES OR REPRESENTATIONS WITH RESPECT TO SUCH SOFTWARE AND PROVIDES NO INDEMNITY FOR SUCH SOFTWARE. IBM GRANTS NO EXPRESS OR IMPLIED PATENT OR OTHER LICENSE WITH RESPECT TO AND IS NOT LIABLE FOR ANY DAMAGES ARISING OUT OF THE USE OF SUCH SOFTWARE.

4.1 Breaking down the scenario

This section outlines the mobile luggage tracking solution of Airline Company A. It describes the process of taking the high-level scenario and breaking it down into smaller pieces. This includes the business process flow within the solution, the new and existing IT components that will be affected, and how these component changes map to the roles in the company's new Mobile division.

Sections 4.2, "Planning and setting up the environments" on page 67 through 4.12, "Analyzing application and adapter usage" on page 191 detail how Airline Company A implemented this mobile application and provides guidance for accomplishing the required tasks. If you want to implement the end-to-end scenario, you must complete each section.

4.1.1 High-level use case

The high-level use case for the mobile luggage tracking solution is shown in Figure 4-1.

A passenger is traveling from one city to another with one stop in between, where the passenger will change airplanes. Upon arrival at the departure airport, the passenger checks in with the airline and checks one bag to be flown as cargo to the passenger's final destination. The bag is tagged with a unique identifier and the pertinent information is entered into the airline's Luggage Tracking Information System (LTIS). However, because of unforeseen problems at Stop 1, the bag is not transferred to the new airplane. When the bag does not arrive at the destination airport, the passenger contacts a nearby customer service representative (CSR) who uses the airline's new mobile application, called LuggageTracker, to locate the missing bag and provide a new estimated arrival time. If the bag, when it arrives, must be delivered to the passenger's home or hotel, the CSR enters the delivery address into the application.

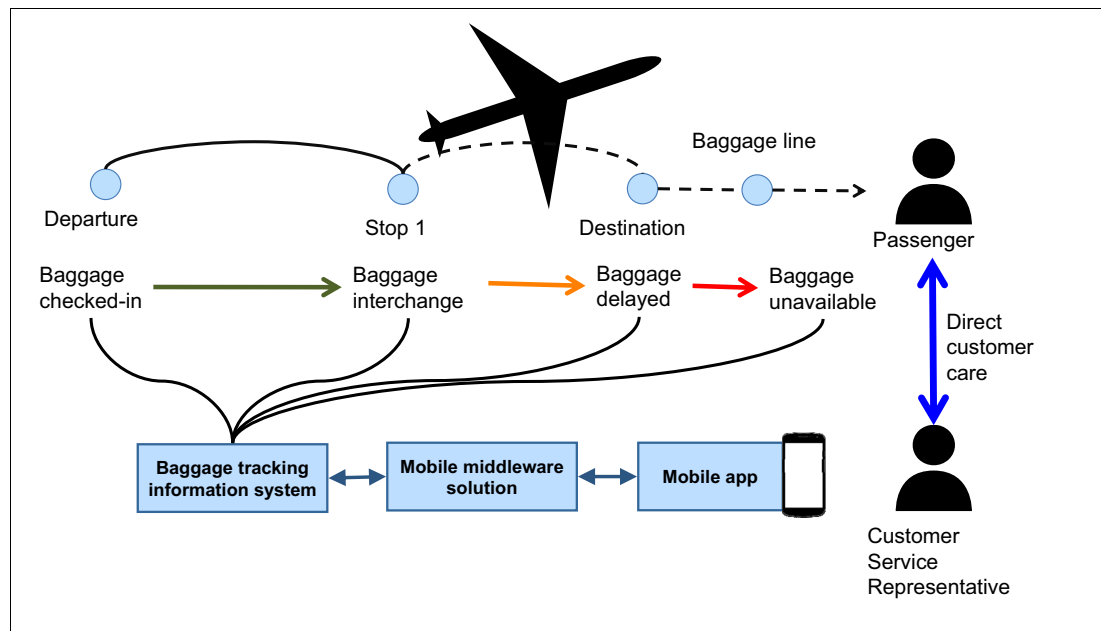


Figure 4-1 High-level use case for the mobile luggage tracking solution

From an IT perspective, IBM Worklight Studio provides the environment to develop the new mobile application. IBM Worklight Server is used as the mobile middleware layer and provides integration to the LTIS and other existing business services.

4.1.2 Agile breakdown and design

Using agile methodologies, the high-level use case is broken down into four main scenarios or *user stories*:

- ▶ **Install:** As a CSR, I need to install the mobile application on my mobile device, if the device is capable of supporting it
- ▶ **Login:** As a CSR, I need to log in to the mobile application on my mobile device
- ▶ **Locate:** As a CSR, I need to use the application to locate a passenger's missing baggage
- ▶ **Deliver:** As a CSR, I need to update the baggage record with a passenger's new or modified delivery address

For each of these user stories, the airline's mobile development and design teams completed a storyboard to define in detail the interactions between the CSR, the mobile application, and the back-end services. These storyboard mock-ups or *flows* are shown in the following subsections.

From a development perspective, each story is broken into tasks that are assigned to appropriate IT staff as discussed in 4.1.4, "Affected roles in Airline Company A's IT organization" on page 67. This task division also involves completing the detailed, lower-level designs and defining the component model.

Install flow

The storyboard mock-up of the Install user story is shown in Figure 4-2 on page 63. It details the initial installation of the application on the mobile device and is a one-time story (because the CSR installs the application only once).

The steps within the Install flow are as follows:

1. The CSR logs into IBM Worklight Application Center (the company's internal application store) using the browser on the mobile device.
2. The CSR installs the Application Center mobile client and enters the authentication credentials (username and password) provided by the IT administrator.
3. The CSR searches the Application Center catalog, finds the LuggageTracker application, and installs it on the mobile device.

After the application is installed, it can be started using the application icon on the mobile device's screen.

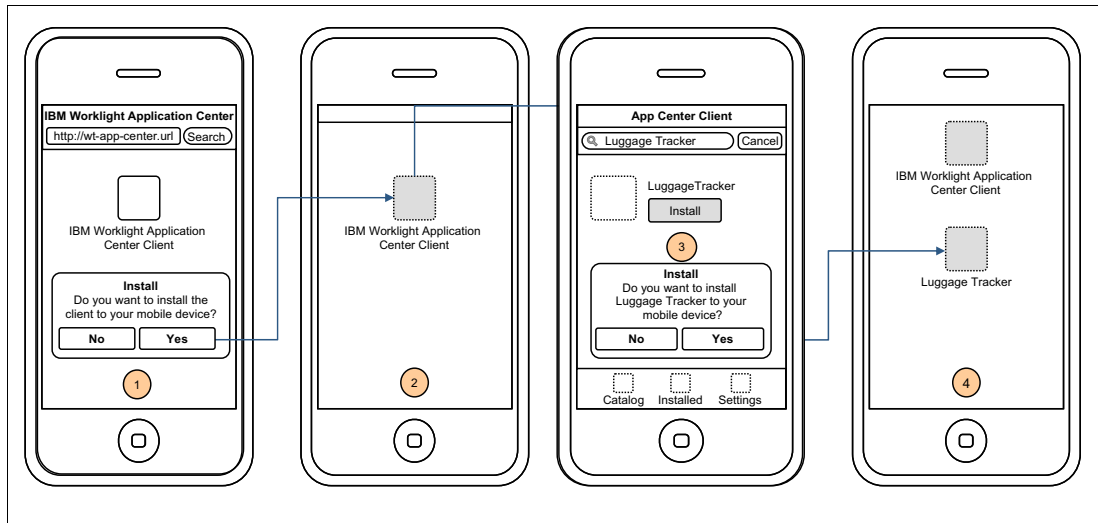


Figure 4-2 Storyboard of Install user story

Login flow

The storyboard mock-up of the Login user story is shown in Figure 4-3. The assumption is that the Install user story has completed.

The steps in the Login flow are as follows:

1. The CSR starts LuggageTracker by clicking its icon on the mobile device screen.
2. A login dialog opens and prompts for authentication. The CSR provides the same credentials that were used to access the application in Application Center prior to installation.
3. The credentials are authenticated against the user registry to ensure access is granted only to authorized personnel.

Upon a successful login, the LuggageTracker home screen is displayed with the options that are available for the CSR to use:

- ▶ Scan Bar Code
- ▶ Retrieve Information
- ▶ Logout

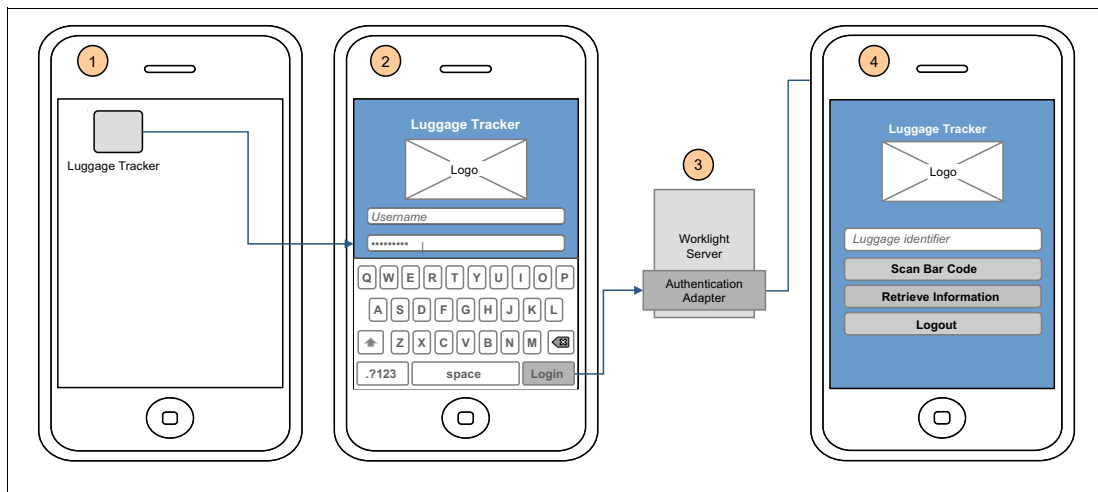


Figure 4-3 Storyboard of Login user story

Locate flow

The storyboard mock-up of the Locate user story is shown in Figure 4-4. It assumes that the Login user story has been completed.

The steps in the Locate flow are as follows:

1. The CSR clicks **Scan Bar Code** in LuggageTracker.
2. The CSR uses the mobile device's camera to scan the bar code on the passenger's luggage receipt or, if scanning fails, uses the device's keypad to manually enter the luggage identifier number.
3. With the luggage tag information successfully scanned or entered, the CSR clicks **Retrieve Information**.
4. Information about the bag is retrieved from the back-end LTIS as follows:
 - Owner's name
 - Current luggage status (such as Delayed or On-time)
 - Current, next, and final destinations
 - Date of last status update
 - Comments previously entered by other CSRs (if any)

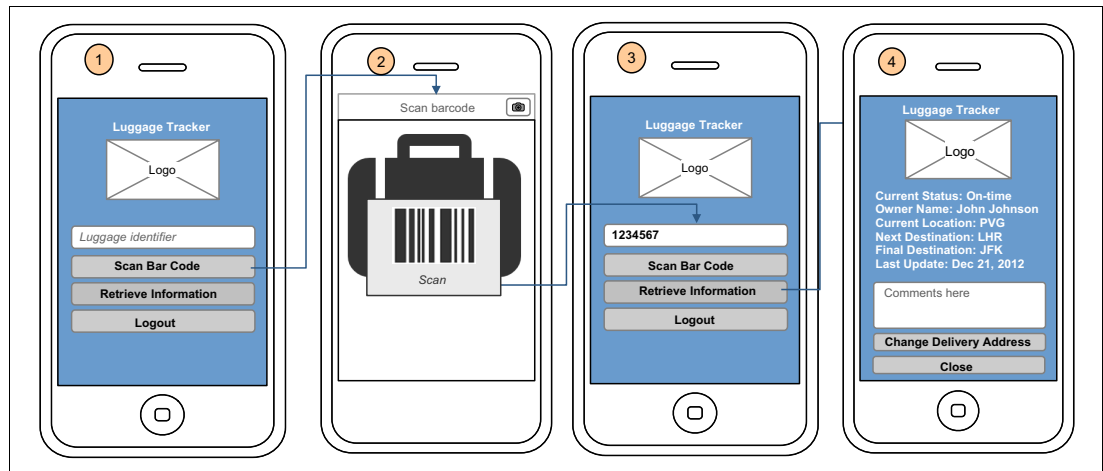


Figure 4-4 Storyboard of Locate user story

Deliver flow

The storyboard mock-up of the Deliver user story is shown in Figure 4-5. It assumes that the Locate user story has completed and that the passenger's delivery address for delayed bags must be updated.

The steps in the Deliver flow are as follows:

1. The CSR clicks **Change Delivery Address** to display a screen where the CSR enters the passenger's preferred delivery address, phone number, and comments.
2. The CSR adds or modifies the delivery address and phone number.
3. The CSR adds any comments, such as the passenger's preferred delivery time, into the Comments field.
4. The CSR clicks **Save and return** to save the information in the LTIS and redisplay the Luggage information screen.

The LTIS record for the bag is now updated and can be used by another CSR who might be asked about the bag's status, or by the delivery service assigned to deliver late-arriving bags to passengers.

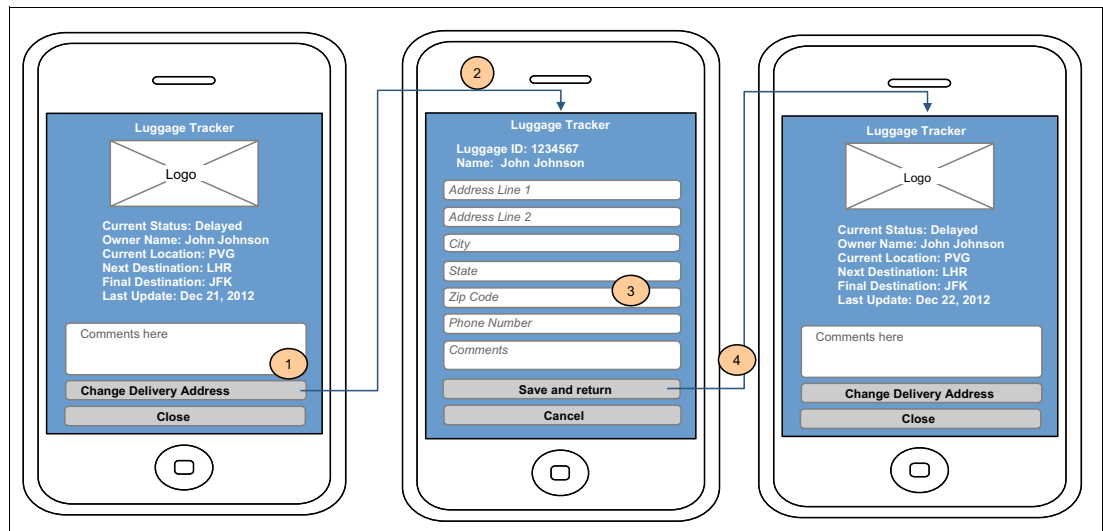


Figure 4-5 Storyboard of Deliver user story

4.1.3 Components of mobile luggage tracking solution

The component model for the airline's luggage tracking solution has three layers:

- Mobile application layer
- Mobile middleware layer
- Business services layer

Within each layer is a set of core components, as shown in Figure 4-6 on page 66. In the figure, the components with dashed borders are components of IBM Worklight Server; those with solid-line borders are custom components written by the airline's IT department.

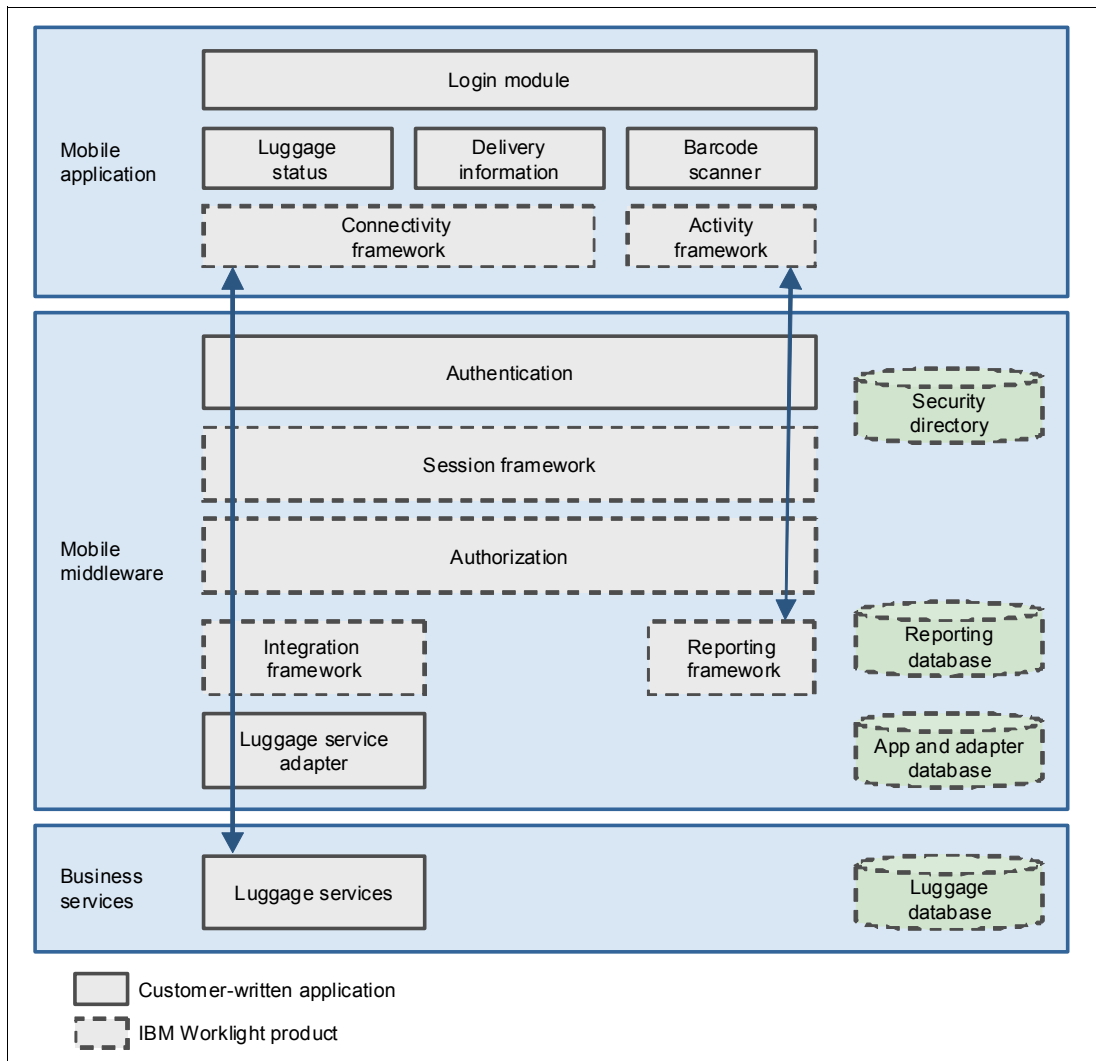


Figure 4-6 Component model of mobile luggage tracking solution

The first layer, the mobile application, resides on the mobile device of a CSR. The mobile application uses the hybrid development approach introduced in 1.3.3, “Development challenges” on page 9 and consists of a login module to verify a user's authentication and authorization (for the Login user story), a luggage status module and bar code scanner (for the Locate user story), and delivery information (for the Deliver user story). It uses two IBM Worklight frameworks in the mobile middleware layer for connectivity and activity reporting.

The second layer, the mobile middleware, is used to secure and simplify communication from LuggageTracker to the back-end LTIS, which is in the business services layer. The mobile middleware layer contains the IBM Worklight Server components and also the adapters developed by the airline's IT staff.

The third layer encompasses the back-end business services. Airline Company A uses existing back-end systems to track luggage, flights, and passengers, so the mobile luggage tracking solution must interface with those systems. To do this, a new web service will be written to connect the mobile middleware layer to the existing systems. This web service has two operations:

- ▶ Retrieve the passenger's luggage information using the unique luggage identifier
- ▶ Update or add the passenger's delivery address as entered by the CSR

4.1.4 Affected roles in Airline Company A's IT organization

Personnel throughout the airline's IT department contribute to the development of LuggageTracker, but the project does not involve *every* possible mobile-related job role outlined in 3.2, "IT department roles" on page 48. The following list describes each involved role and explains how the company's mobile organization collaborates to complete the solution:

- ▶ Web mobile application developer

The login module, luggage status query tool, and delivery information updater will be implemented with web technology because they do not require native device functions. A web mobile application developer uses IBM Worklight Studio to create these parts of the mobile application.

- ▶ Native mobile application developer

The bar code scanner cannot be implemented with web technology because it requires access to a mobile device's camera. So a native mobile application developer will implement this module, using a Worklight shell. The shell will then be combined with the other components of the application.

- ▶ Mobile server developer; integration developer

A mobile server developer will implement an IBM Worklight adapter to retrieve luggage status information and update luggage delivery information. The adapter validates the request and, if it contains a valid luggage identifier number or delivery address, will call a back-end business service to retrieve the luggage status or update the delivery address. These business services are then published by an integration developer.

- ▶ Mobile server developer

To ensure that the mobile application functions are available only to authorized CSRs, a mobile server developer implements a security component that permits only authorized communications between the mobile application and the back-end services.

- ▶ Mobile application tester

To perform functional testing, a mobile application tester installs LuggageTracker by downloading it from IBM Worklight Application Center to his or her mobile device and then testing the various functions of the application.

- ▶ System administrator

To keep LuggageTracker running, a system administrator maintains and monitors the IBM Worklight infrastructure (the Mobile Enterprise Application Platform (MEAP) that is selected by Airline Company A in 3.6, "Selecting a Mobile Application Platform" on page 54). This includes making the mobile application available in the company's internal application store and making updates to the application available to the CSRs.

4.2 Planning and setting up the environments

Airline Company A has three mobile application environments:

- ▶ Development
- ▶ Pre-production
- ▶ Production

The development environment is used for both development and unit testing, including testing with the device simulators that are part of IBM Worklight Studio.

The pre-production environment mimics the production environment but without any high-availability mechanisms. It is used by mobile application testers to evaluate the functionality of the new application using real mobile devices, and hosts the alpha and beta versions of the application that are tested by selected CSRs. These alpha and beta versions are distributed, with strict controls, through IBM Worklight Application Center.

The production system is the one used by CSRs in the field as they interact with passengers. This production system is designed with multiple high-availability mechanisms to ensure 24x7 availability of the luggage tracking solution.

For the purposes of this book, only the development and pre-production environments were created.

4.2.1 Setting up the development environment

The development environment consists of all necessary systems and tooling required for implementing the solution, based on the specific role in the IT organization's Mobile division.

This section describes what you need to set up your development environment.

System developers

System developers implement the business logic of the application, which does not require any particular mobile-related software. Instead, they need a development environment for writing, debugging, and testing Java-based web services, including application servers.

The following products must be installed and configured for system development:

- ▶ The Eclipse development environment (either a stand-alone version or the version that is included with IBM Worklight Studio), which is used for development and unit testing. For details about installing Worklight Studio, see the following locations, depending on your development operating system: 6.3.1, "Installing IBM Worklight Studio on Mac OS X" on page 301 or 6.3.3, "Installing IBM Worklight Studio on Windows XP" on page 308.
- ▶ WebSphere Application Server Liberty Profile, which is used as the application server for unit testing the Java-based web services. For installation guidance, see 6.4, "Installing WebSphere Application Server Liberty Profile" on page 309.

Mobile server developers

Mobile server developers create and test the adapters that are deployed to the Worklight server and act as the integration point between the mobile application and back-end services.

The following product must be installed and configured for mobile server development:

- ▶ IBM Worklight Studio, which is used for development and unit testing of the adapters. For details about installing Worklight Studio, see these locations, depending on your development operating system: 6.3.1, "Installing IBM Worklight Studio on Mac OS X" on page 301 or 6.3.3, "Installing IBM Worklight Studio on Windows XP" on page 308.

Web mobile application developers

Web mobile application developers create and test the user interface of mobile applications that are not platform-specific.

The following product must be installed and configured for web development:

- ▶ IBM Worklight Studio, which is used for development and unit testing of an application's user interface. For installation details, see the following locations, depending on your development operating system: 6.3.1, "Installing IBM Worklight Studio on Mac OS X" on page 301 or 6.3.3, "Installing IBM Worklight Studio on Windows XP" on page 308.

Native mobile application developer

Native mobile application developers create and test the user interface of mobile applications that target specific platforms and use device-specific features (such as the camera).

The following products must be installed and configured for native development targeting the Android platform:

- ▶ IBM Worklight Studio, which is used for development and unit testing of platform-specific user interfaces. For details about installing Worklight Studio, see these locations, depending on your development operating system: 6.3.1, "Installing IBM Worklight Studio on Mac OS X" on page 301 or 6.3.3, "Installing IBM Worklight Studio on Windows XP" on page 308.
- ▶ The Android Software Development Kit (SDK) on Apple OS X, which is used when a native developer wishes to do Android builds of a mobile application using an Apple-based PC. For installation details, see 6.3.2, "Installing the Android SDK on Mac OS X" on page 303.

The following products must be installed and configured for native development targeting the iOS platform:

- ▶ IBM Worklight Studio on Mac OS X, which is used for development and unit testing of platform-specific user interfaces. For installation guidance, see 6.3.1, "Installing IBM Worklight Studio on Mac OS X" on page 301.
- ▶ Apple iOS Development tools, which are used to build iOS applications. To set up and configure these tools, see *Start Developing iOS Apps Today* at the following location:
<https://developer.apple.com/library/ios/#referencelibrary/GettingStarted/RoadMapiOS/chapters/GetToolsandInstall.html>

4.2.2 Setting up the pre-production environment

The pre-production environment consists of all necessary systems and software required to run and maintain the solution.

For Airline Company A's luggage tracking application, the pre-production environment uses the topology shown in Figure 4-7 on page 70. The environment consists of the IBM Worklight Server, a business services server, and simulated back-end systems. The Worklight server accesses the business services, which then access the simulated back-end systems.

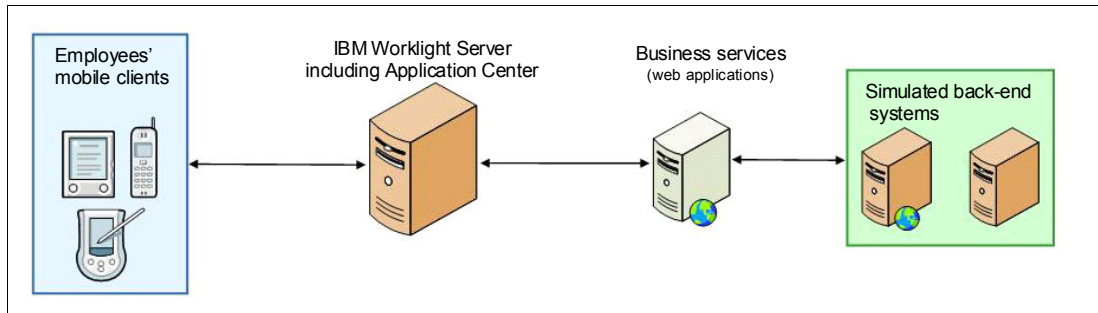


Figure 4-7 Pre-production environment model for luggage tracking solution

For simplicity, the topology assumes that every employee's mobile device has access to the Worklight server and IBM Worklight Application Center. In a real application scenario, a solution such as IBM Endpoint Manager for Mobile Devices is incorporated to force employees to register their mobile devices and then push the necessary network configuration to each registered device. With this approach, the mobile devices of the employees can be registered with the IT infrastructure as endpoints, and then they can access Worklight Application Center to download LuggageTracker.

This section describes what you need to set up your pre-production environment.

System administrator

The system administrator deploys the mobile application and Worklight adapters to the Worklight server, and deploys the mobile applications to Application Center, which acts as the company's internal application store. To accomplish this in the pre-production environment, the following software must be installed and configured:

- ▶ IBM Worklight Server, which acts as the MEAP for Airline Company A. For guidance on installing IBM Worklight Server, see 6.2, "Installing IBM Worklight Server" on page 295.
- ▶ IBM WebSphere Application Server Liberty Profile, which runs the web applications that simulate the existing back-end business services and provides the integration layer between the back-end services and IBM Worklight Server. For installation instructions, see 6.4, "Installing WebSphere Application Server Liberty Profile" on page 309.

Mobile application testers

For quality assurance, pre-production mobile application testers need both Android and iOS mobile devices and must have access to Worklight Application Center. The testers will use the following product:

- ▶ The IBM Worklight Application Center mobile client, which is installed on each mobile device to enable them to download LuggageTracker from the company application store. For details about installing and configuring the mobile client, see 4.11, "Using feedback mechanisms" on page 189.

4.3 Providing enterprise services

As described in 4.1.1, “High-level use case” on page 61, LuggageTracker will interact with the airline’s existing back-end services to retrieve a bag’s current location from LTIS and, if necessary, add or update the passenger’s delivery address.

When existing applications already provide a function that is needed for a mobile application, an integration layer can be introduced to use the existing functions as enterprise services. Depending on the enterprise architecture, possible approaches include implementing a web services integration layer or using an enterprise service bus (ESB).

To illustrate the integration capabilities of IBM Worklight Studio for this book, an integration layer was used consisting of two Representational State Transfer (RESTful) web services deployed to an IBM WebSphere Application Server Liberty Profile application server.

Figure 4-8 shows the main components and flow of data from the mobile application, through the integration layer, to the back-end services.

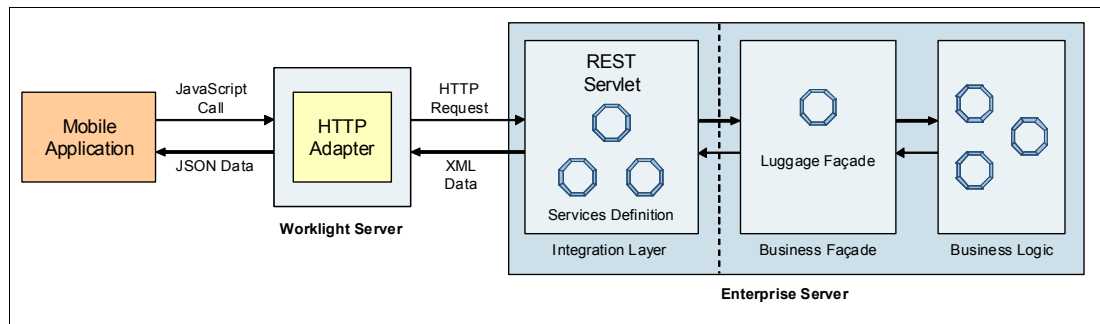


Figure 4-8 Data flow from the application to the back-end services

As shown in the figure, the mobile application first uses a JavaScript call to invoke the HTTP adapter that is responsible for calling the back-end services. For more information about HTTP adapters, see 4.7, “Creating the adapters” on page 88.

After the adapter call, an HTTP request is generated to invoke the services exposed in the integration layer. The Java API for RESTful Web Services (JAX-RS) servlet in this layer calls the existing back-end services (depicted here in the Business Façade and Business Logic blocks) and handles the conversion between the Java objects that are returned from the back-end services and the XML format that is sent back to the adapter. The Façade is a helper object (using the facade design pattern common in software development) that provides a simplified interface to the back-end business logic.

The following sections describe the main components of the integration layer in more detail.

4.3.1 Services definition

In the integration layer, the REST servlet defines two services that the HTTP adapter can call using an HTTP request:

- ▶ Retrieving luggage information
- ▶ Updating luggage information

These services are implemented using JAX-RS annotations. Requests are invoked using the following URL:

`http://<server>:<port>/<app_root>/<service_prefix>/<service_name>/<parameters>`

The variables in the invocation URL are as follows:

server	The server IP address or host name.
port	The server port listening for HTTP requests.
app_root	The web application context root. In this scenario, it is AirlineREST.
services_prefix	The path used to indicate the root location of JAX-RS services. In this scenario, it is /jaxrs/.
service_name	The name of the service to be invoked, usually representing an entity. In this scenario, only the luggage service is represented.
parameters	Any additional parameters needed to invoke the service. In this scenario, the luggage identifier number is used.

Retrieving luggage information

On a GET request, the luggage service returns the information for the specified luggage tracking identifier. The service expects the luggage tracking identifier as input and returns the luggage object in XML format. The complete luggage service specification is shown in Table 4-1.

Table 4-1 Service specification for luggage service on GET request

URL	<code>http://server:port/AirlineREST/jaxrs/luggage/{trackingId}</code>
Method	GET
Input	trackingId: luggage tracking identifier
Output	luggage XML: the luggage object

Updating luggage information

On a POST request, the luggage service updates the passenger's delivery address and any comments entered by a CSR for a given piece of luggage. The service specification is shown in Table 4-2. The service expects the luggage tracking identifier, address information, and comments as input and returns the updated luggage object in XML format.

Table 4-2 Service specification for luggage service on POST request

URL	<code>http://server:port/AirlineREST/jaxrs/luggage</code>
Method	POST
Input	<ul style="list-style-type: none"> ▶ trackingId: luggage tracking identifier ▶ name: delivery recipient name ▶ addressLine1: first line of delivery address ▶ addressLine2: second line of delivery address ▶ city: delivery city ▶ state: delivery state ▶ zipCode: postal code ▶ phoneNumber: contact phone number ▶ comments: additional delivery comments
Output	luggage XML: the updated luggage object

4.3.2 XML Data Format

As mentioned in the service specifications, the services return a luggage object in XML format. This XML format is used to exchange information between the various applications throughout the enterprise. The XML definition for a tracked piece of luggage within the LTIS is shown in Example 4-1.

Example 4-1 Luggage entity XML definition

```
<luggage>
  <trackingId> ... </trackingId>
  <ownerName> ... </ownerName>
  <status> ... </status>
  <flightNumber> ... </flightNumber>
  <currentLocation> ... </currentLocation>
  <nextLocation> ... </nextLocation>
  <finalDestination> ... </finalDestination>
  <deliveryAddress>
    <name> ... </name>
    <addressLine1> ... </addressLine1>
    <addressLine2> ... </addressLine2>
    <city> ... </city>
    <state> ... </state>
    <zipCode> ... </zipCode>
    <phoneNumber> ... </phoneNumber>
    <comments> ... </comments>
  </deliveryAddress>
  <comments> ... </comments>
  <lastUpdate> ... </lastUpdate>
</luggage>
```

The marshalling and unmarshalling of the luggage object in XML format is implemented with the Java Architecture for XML Binding (JAXB) specification, using the class shown in Example 4-2.

Example 4-2 JAXB specification for luggage object

```
import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement(name="luggage")

public class Luggage {
  private String trackingId;
  private String ownerName;
  private String status;
  private String flightNumber;
  private String currentLocation;
  private String nextLocation;
  private String finalDestination;
  private Address deliveryAddress;
  private String comments;
  private String lastUpdate;
  ...
}
```

4.3.3 Services implementation

The RESTful services are implemented in a Java EE web application, using a Java interface to define public methods and the required JAX-RS annotations. The business processes are invoked using a concrete Java class that implements this interface. Annotations can be used directly in the concrete class, but this methodology provides greater flexibility and code readability.

The LuggageService Java interface class, which is shown in Example 4-3, is used to define the JAX-RS service specification.

Example 4-3 LuggageService Java interface

```
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.xml.bind.JAXBElement;
import com.ibm.itso.saw210.beans.Luggage;

@Path("/luggage")

public interface LuggageService {
    @GET
    @Path(value="/{luggageId}")
    @Produces(MediaType.APPLICATION_XML)
    public Luggage getLuggageInformation(@PathParam(value="luggageId") String
luggageId);

    @POST
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    @Produces(MediaType.APPLICATION_XML)
    public Luggage updateDeliveryInformation(
        @FormParam(value="trackingId") String trackingId,
        @FormParam(value="name") String name,
        @FormParam(value="addressLine1") String addressLine1,
        @FormParam(value="addressLine2") String addressLine2,
        @FormParam(value="city") String city,
        @FormParam(value="state") String state,
        @FormParam(value="zipCode") String zipCode,
        @FormParam(value="phoneNumber") String phoneNumber,
        @FormParam(value="comments") String comments) ;
}
```

The LuggageServiceImpl class, which is shown in Example 4-4 on page 75, is the concrete implementation class of the interface in the previous example. It calls the corresponding methods in the façade (shown in Figure 4-8 on page 71) to invoke the business logic with the supplied parameters.

Example 4-4 Concrete Java implementation of the LuggageService interface

```
import javax.ws.rs.Path;

import com.ibm.itso.saw210.beans.Address;
import com.ibm.itso.saw210.beans.Luggage;
import com.ibm.itso.saw210.facade.LuggageFacade;

@Path("/luggage")
public class LuggageServiceImpl implements LuggageService {

    public Luggage getLuggageInformation(String luggageId) {
        return LuggageFacade.getLuggageInformation(luggageId);
    }

    public Luggage updateDeliveryInformation(
        String luggageId,
        String name,
        String addressLine1,
        String addressLine2,
        String city,
        String state,
        String zipCode,
        String phoneNumber,
        String comments) {

        return LuggageFacade.updateDeliveryInformation(
            String luggageId,
            String name,
            String addressLine1,
            String addressLine2,
            String city,
            String state,
            String zipCode,
            String phoneNumber,
            String comments);
    }
}
```

4.4 Planning for security

The authentication framework of IBM Worklight ensures that access to Airline Company A's luggage tracking application is secure. This includes not only access to the application itself, but also proper access to the back-end resources that the application uses. The following subsections describe the application's authentication model and the client-side and server-side components that are involved in the end-to-end security solution.

4.4.1 Understanding the mobile authentication flow

The application uses the IBM Worklight authentication framework to ensure secure access to application functions and to the airline's back-end services, similar to the security requirements when a CSR uses the existing systems to check the status of delayed luggage. The interaction between the client (the mobile application) and the server, which contains the company's authentication adapter, is depicted in Figure 4-9 on page 76.

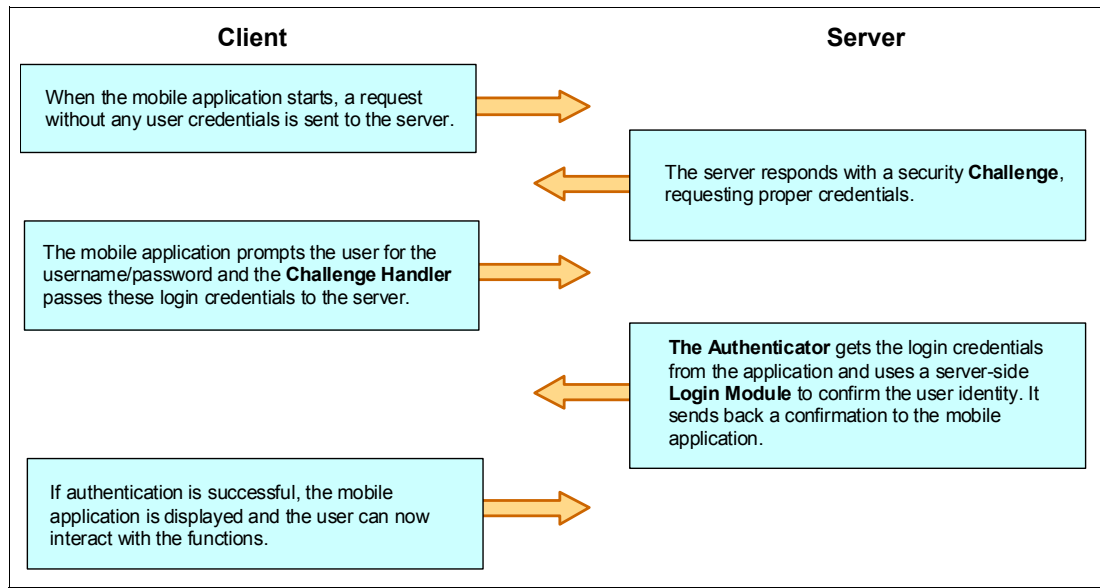


Figure 4-9 Authentication process to authorize access to LuggageTracker

As you see, the authentication process uses components on both the client and server sides of the application. This process is defined in an *authentication realm* in the Worklight server's configuration, and consists of a *Challenge Handler* on the client side, and an *Authenticator* and a *Login Module* on the server side.

The following sections give more detail about each of these components.

4.4.2 Planning client-side authentication

On the client-side, when a request to use LuggageTracker must be authenticated, including when the application is first started, a login screen is displayed to gather the user's credentials while all other screens in the application are hidden. When the user has been validated (which occurs on the server side), the login screen disappears and the application's main screen is displayed (see Figure 4-10).

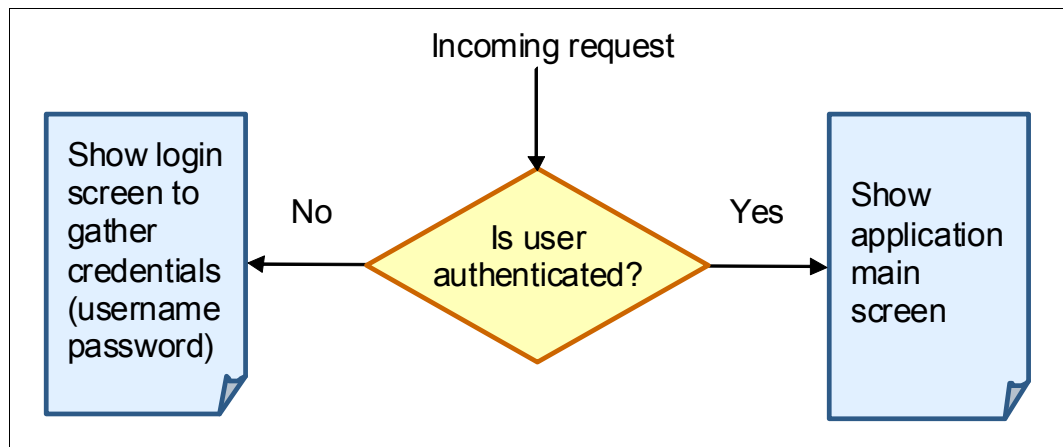


Figure 4-10 Client-side authentication process

The client-side components used for authentication are shown in Figure 4-11. The login button is part of the login screen and initiates the submission of the user's credentials for authorization. The credentials are accepted by the Challenge Handler (defined in the `auth.js` file) and then sent to the server-side components for validation against the user registry.

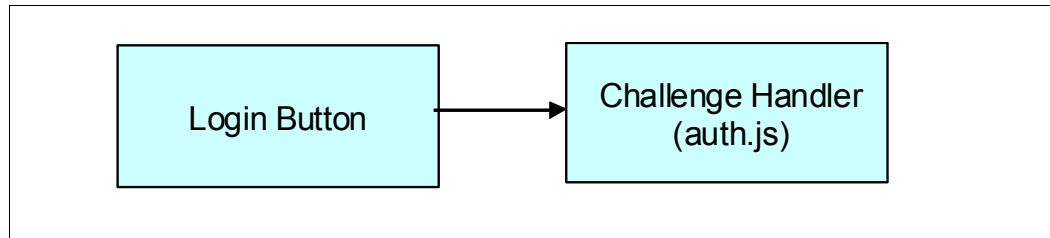


Figure 4-11 Client-side authentication components

4.4.3 Planning server-side authentication

On the server side, authentication for LuggageTracker is validated using an IBM Worklight adapter. Other authentication mechanisms were considered, but this one was selected for the following reasons:

- ▶ Uses the IBM Worklight Server authentication framework.
- ▶ Implements authentication logic using JavaScript, which is easily written and tested using skills that already exist in Airline Company A's IT department.
- ▶ Allows authentication methods to be changed by altering the server-side adapter alone, with no changes required within the client-side application.

The server-side components of the authentication process (Figure 4-12) consist of a login module, the authenticator (adapter), and the user registry.

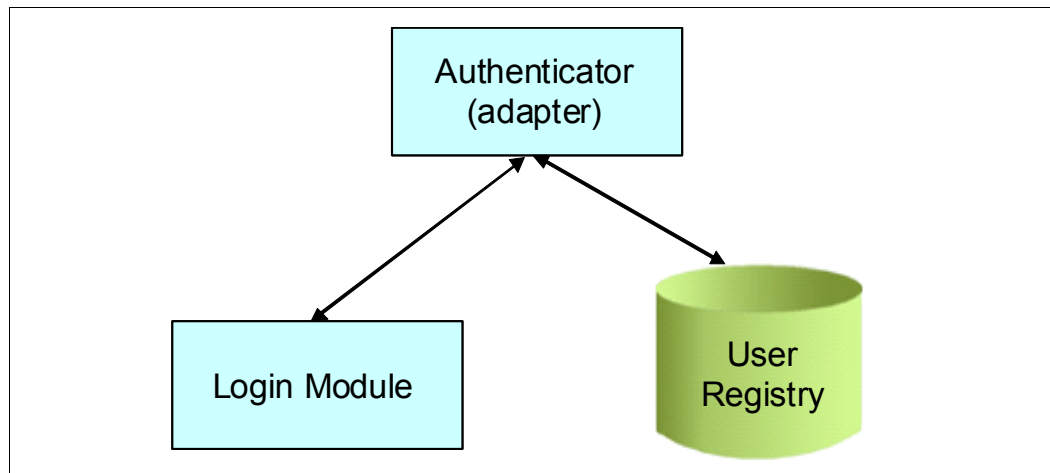


Figure 4-12 Server-side authentication components

The figure shows the server-side components of the authentication realm. The challenge handler on the client side collects and passes the user credentials to the authenticator. The authenticator and login module take those credentials and validate them against the company's user registry. The login module also creates a *user identity* object that holds the user properties for the duration of the session. When the user logs out or a session timeout occurs, the login module destroys this user identity object. The login module can also be used to record login attempts for audit purposes if necessary.

For more information about Worklight adapter-based authentication, go to the following address:

ftp://public.dhe.ibm.com/software/mobile-solutions/worklight/docs/v505/Module_22_-_Adapter_Based_Authentication.pdf

4.4.4 Pulling it all together

The complete authentication model for LuggageTracker is depicted in Figure 4-13. The individual components are implemented later in this chapter.

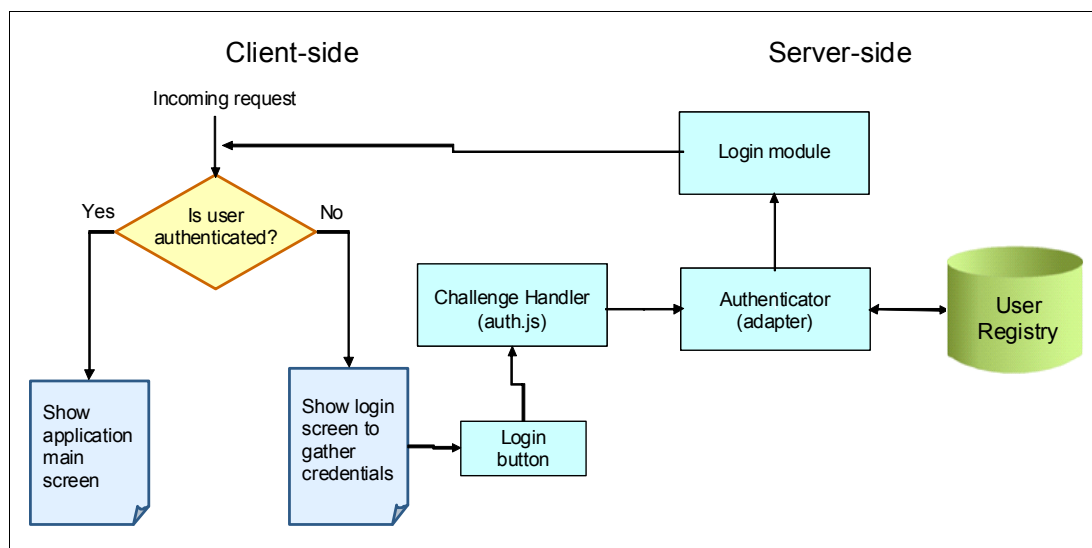


Figure 4-13 End-to-end authentication flow

4.5 Operating the Mobile Enterprise Application Platform

Typically, a system administrator is responsible for providing and operating the IBM Worklight portion of the Mobile Enterprise Application Platform (MEAP), managing access to the MEAP, and generating reports about its use. The administrator's job also includes deploying and updating the application and Worklight adapters on the Worklight server, and deploying and updating the applications in Worklight Application Center, which serves as the corporate application store.

The system administrator is also responsible for configuring and running reports and analytics to measure the load on the MEAP and evaluate usage patterns of the applications that are deployed there. By reviewing these usage patterns, the system administrator can detect increasing load as it becomes apparent and scale up the Worklight server infrastructure to handle more work.

4.5.1 Managing access to the Application Center

Managing access to the applications is a two-phase process:

1. You address access to Application Center and the Application Center mobile client (which is explained in this section)
2. You address access to the applications and specific application versions that are being offered (explained in 4.5.2, "Managing access to mobile applications" on page 81).

The *security role* associated to a user is based on the group or groups that the user is a member of, and determines the user's level of access to Application Center. Application Center has two Java Platform, Enterprise Edition (JEE) security roles defined:

- ▶ **appcenteruser**: Allows access to Application Center through the Application Center mobile client only
- ▶ **appcenteradmin**: Allows a user to perform administrative tasks using Application Center

The CSRs who will use LuggageTracker must have the **appcenteruser** role, so that they can use the mobile client to install the application. Granting access to the Application Center mobile client involves the following tasks:

1. Create the users.
2. Add the users to a group.
3. Assign the **appcenteruser** role to that group.

Airline Company A chose to use only the basic repository that is included with the Liberty Profile, so the new users and groups were added to the Liberty Profile's `server.xml` file within the basic registry named `applicationcenter-registry`.

Creating the users

To create the users, use the following steps:

1. Open the `<wl_install_dir>/server/wlp/usr/servers/worklightServer/server.xml` file (where `<wl_install_dir>` is the directory where the Worklight Server is installed).

In the file, locate the following line:

```
<basicRegistry id="applicationcenter-registry" realm="ApplicationCenter">
```

2. Find the line that defines the user named `demo`, then add lines below it for your new users, providing a name and password for each, as shown in Example 4-5.

Example 4-5 Adding new user definitions

```
<basicRegistry id="applicationcenter-registry" realm="ApplicationCenter">
  <user name="appcenteradmin" password="admin"/>
  <!-- The other users have normal privileges. -->
  <user name="demo" password="demo"/>
  <user name="CSR1" password="csr1"/>
  <user name="CSR2" password="csr2"/>
  <user name="CSR3" password="csr3"/>
  <user name="CSR4" password="csr4"/>
  <user name="CSR5" password="csr5"/>
  <user name="CSR6" password="csr6"/>
  <user name="Tester1" password="tester1"/>
  <user name="Tester2" password="tester2"/>
  ....
  ....
</basicRegistry>
```

Adding the users to a group

Now that the individual user authentications (user name and password) are defined, the users must be associated with the proper security role (**appcenteruser**). Although adding individual users to a security role is possible, in this scenario the users are members of a group and that group will be associated to the security role.

To add users to a group and associate that group to the appcenteruser security role, use the following steps:

1. In the same `server.xml` file where you added your users, find the line that defines the group named `appcentergroup`. This group is for users that can run administrative tasks in Application Center, and already contains two users (`appcenteradmin` and `demo`) that were created during the installation process.
2. Immediately beneath the definition for `appcentergroup`, add a new group named `appcenterusergroup`, to contain the user IDs for the CSRs and testers, as shown in Example 4-6.

Example 4-6 Defining the new appcenterusergroup group and defining the group members

```
<basicRegistry id="applicationcenter-registry" realm="ApplicationCenter">
    ....
    ....
    <group name="appcentergroup">
        <member name="appcenteradmin"/>
        <member name="demo"/>
    </group>
    <group name="appcenterusergroup">
        <member name="CSR1"/>
        <member name="CSR2"/>
        <member name="CSR3"/>
        <member name="CSR4"/>
        <member name="CSR5"/>
        <member name="CSR6"/>
        <member name="tester1"/>
        <member name="tester2"/>
    </group>
</basicRegistry>
```

Assigning the appcenteruser role to the group

Finally, the `appcenterusergroup` group must be associated to the security role `appcenteruser`, which allows the group members to use the Application Center mobile client to search for and install applications.

Use the following steps to add the proper security role to the new group:

1. In the same `server.xml` file, find the definition of the IBM Application Center application (which is shown at the start of Example 4-7) and locate the definition of the `appcenteradmin` security role. Beneath that security role definition, add the 3 lines of HTML in Example 4-7 to define the `appcenteruser` security role and associate the `appcenterusergroup` to that role.
2. Save the `server.xml` file.

Example 4-7 Adding a security role for appcenterusergroup

```
<!-- Declare the IBM Application Center application. -->
<application id="applicationcenter" name="applicationcenter"
    location="applicationcenter.war" type="war">
    <application-bnd>
        <security-role name="appcenteradmin">
            <group name="appcentergroup"/>
        </security-role>
```

```
<security-role name="appcenteruser">
  <group name="appcenterusergroup"/>
</security-role>
</application-bnd>
</application>
```

At this point, you have configured Application Center so that the CSRs and Testers can access the mobile client, search for applications they have access to, and install them.

4.5.2 Managing access to mobile applications

Airline Company A decided to create two versions of the luggage tracking application:

- ▶ An early adopter version, LuggageTrackerBeta
- ▶ The production version, LuggageTracker

Access to these application versions will be controlled through Application Center groups. Users with access to only the production version will belong to an Application Center group named CSR. Users with access to both the production version and the beta version will belong to an Application Center group named CSR-Early Adopters. There will also be a test team with access to both versions; this Application Center group will be named Testers.

The eight users that were defined in the previous section are assigned to these groups as follows:

- ▶ CSR1 and CSR2 in the CSR-Early Adopters group
- ▶ CSR3, CSR4, CSR5, and CSR6 in the CSR group
- ▶ Tester1 and Tester2 in the Testers group

The steps for managing access to mobile applications are similar to those needed to manage access to Application Center:

1. Create the Application Center groups.
2. Add the user or users to a group.
3. Assign the appcenteruser role to that group.

The first two steps are accomplished in this section, and the final step will be completed when the application version is deployed (4.9, “Deploying the application” on page 170).

Creating the Application Center groups

To create the three Application Center groups, use the following steps:

1. From Worklight Server, open Application Center in a browser with the following URL:
`http://localhost:9080/applicationcenter`
2. Log in using the appcenteradmin user ID and password (as defined in the `server.xml` file). If you did not change the password, it defaults to admin.
3. Click the **Users / Groups** link at the top of the Application Center interface.
4. Click **Create Group** to display the Create Group window shown in Figure 4-14 on page 82.

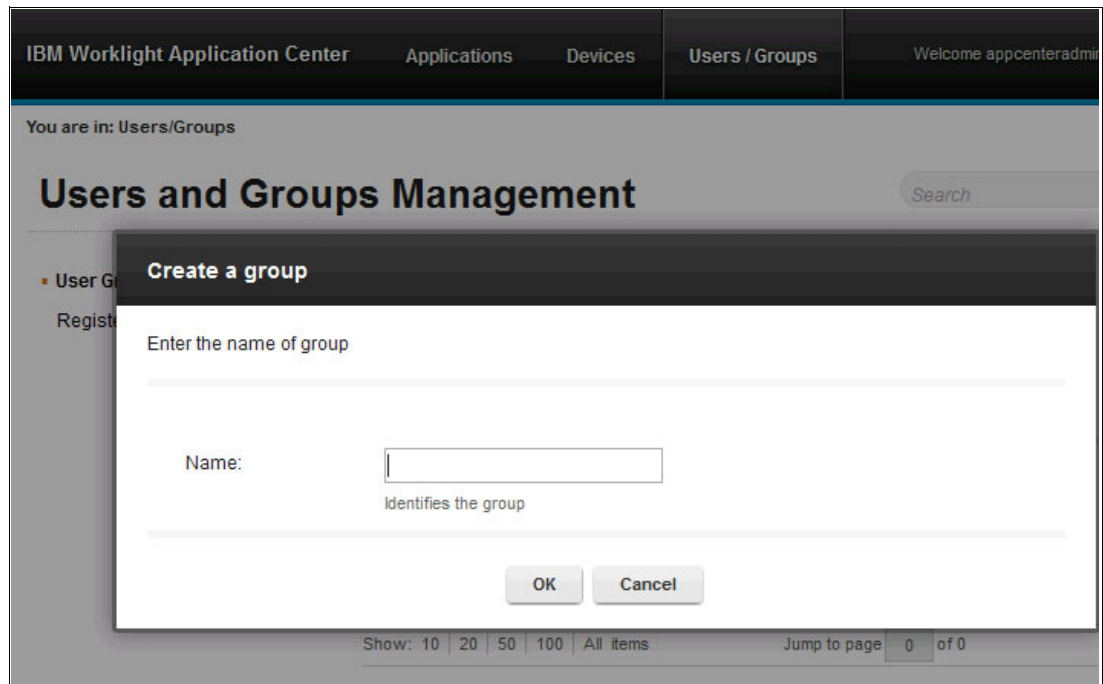


Figure 4-14 Creating a group in Application Center

5. Enter CSR in the Name field and click **OK**. The new group is displayed in the list on the User Groups page.
6. Repeat Steps 4 on page 81 and 5 to create the CSR-EarlyAdopters and Testers groups.
7. Verify that all three groups are shown in the group list, as shown in Figure 4-15.

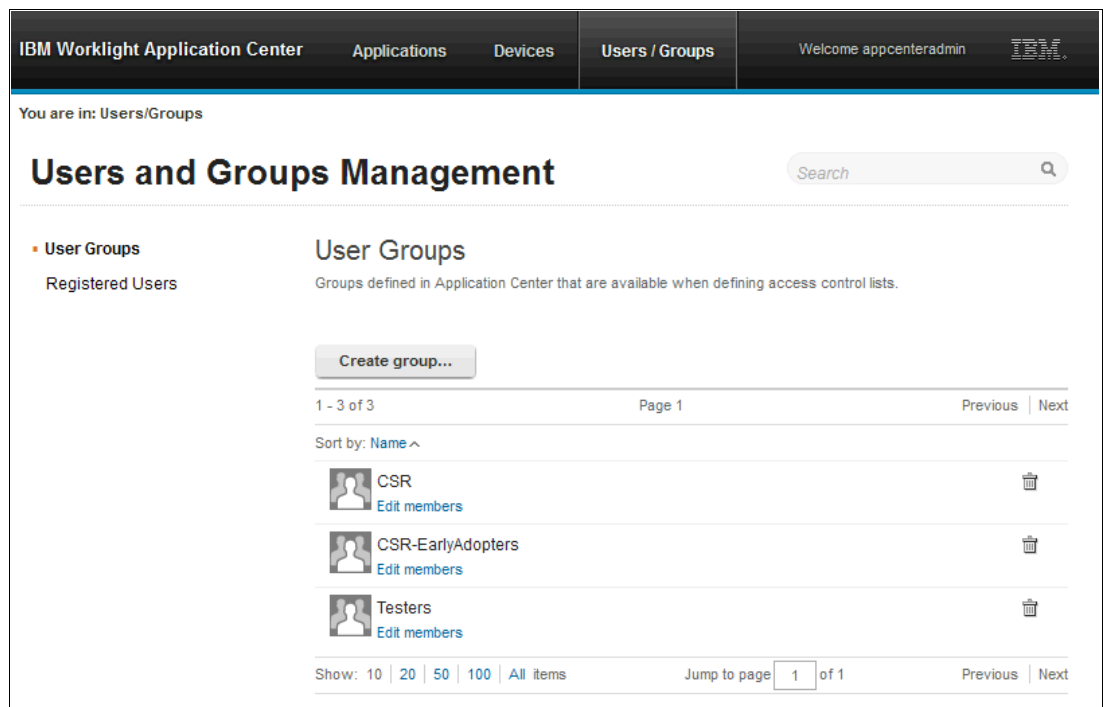


Figure 4-15 Groups shown in the User Groups list

Application Center maintains the group and user directory in its own database. New users are automatically added to the database the first time they log in to the Application Center. However, if a company does not allow unrestricted access to applications, the new user is able to log in but does not have any authorized applications to install. So to ensure that users see the list of authorized applications the first time they log in, you must add them manually to the database. Users that are automatically created the first time they log in can be added to a group later.

To manually add the users and assign them to the appropriate group, use the following steps:

1. From the list of user groups, click the **Edit Members** link located below a particular group, such as CSR, to display the group's Group Membership page.
2. Type the user name CSR3 into the text field and click **Add**.
3. Because that user does not exist yet in the Application Center user registry, the Register a user dialog is displayed. Leave the Name as CSR3 but change the Full Name to anything you want (in this example, **John Doe**), and then click **OK**.
4. Repeat Steps 2 and 3 for CSR4, CSR5, and CSR6, giving each a full name if you want.

When finished, the members are listed on the Group Membership page, as shown in Figure 4-16.

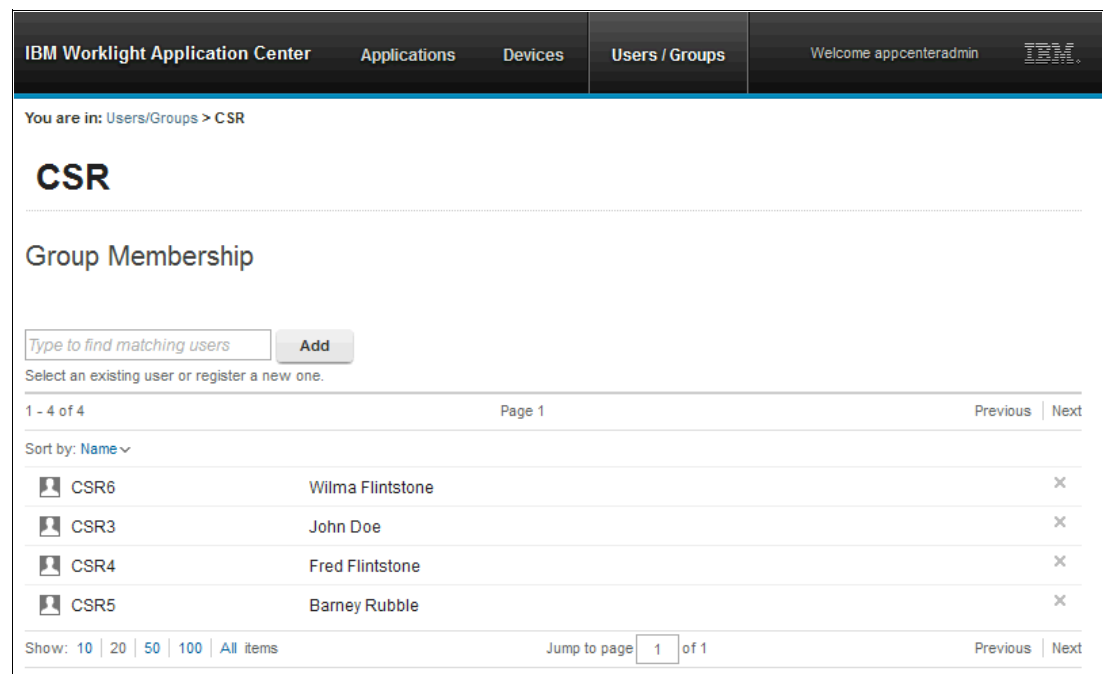


Figure 4-16 CSR group with four members

Repeat this process to add the remaining four users to their associated groups:

- CSR1 and CSR2 to the group CSR-EarlyAdopters
- Tester1 and Tester2 to the group Testers

You are finished creating the users and groups for LuggageTracker and LuggageTrackerBeta. The complete list of users is presented in the Registered Users section of the Users and Groups screen, as shown in Figure 4-17 on page 84.

IBM Worklight Application Center
Applications
Devices
Users / Groups
Welcome appcenteradmin
IBM

You are in: Users/Groups

Users and Groups Management

Search

User Groups

Registered Users

Registered Users

Registered users of the Application center: mobile users, console users, local group members or members of access control list.

Register user...


















1 - 9 of 9

Page 1

Previous

Next

Sort by: Name ^

 appcenteradmin	appcenteradmin	
 CSR5	Barney Rubble	
 CSR1	CSR1	
 CSR2	CSR2	
 CSR4	Fred Flintstone	
 CSR3	John Doe	
 Tester1	Tester1	
 Tester2	Tester2	
 CSR6	Wilma Flintstone	

Show: 10 | 20 | 50 | 100 | All items

Jump to page 1 of 1

Previous

Next

Figure 4-17 List of all registered users

The users are now registered with Application Center. This enables them to login using the mobile device's browser and install the mobile client, which you deploy in the next section.

4.5.3 Deploying the Worklight Application Center mobile client

The Application Center mobile client, sometimes referred to as the Application Center installer, is an application that is installed on Android and iOS devices and allows prospective users to view available applications in the company's internal application store and select one or more applications to download and install on their mobile device.

For the Android platform, the mobile client is provided with IBM Worklight in a ready-to-deploy Android application package (APK) file, named `IBMApplicationCenter.apk`. This file can be deployed by the system administrator without any modifications.

In contrast, the iOS version of the mobile client must be compiled into an iPhone application archive (IPA) file and signed by using Xcode. This step is completed by a native mobile application developer, after which the file is deployed by the system administrator.

To build the mobile client application, IBM Worklight provides a Worklight Studio project. This project is located in the `<wl_install_dir>/ApplicationCenter/installer` directory (where `<wl_install_dir>` is the location where the Worklight Server was installed).

The steps for importing the project into Worklight Studio and building the mobile client application are in the IBM Worklight Information Center:

http://pic.dhe.ibm.com/infocenter/wrklght/v5r0m5/index.jsp?topic=%2Fcom.ibm.worklight.help.doc%2Fappcenter%2Fr_ac_prereqs_build_client.html

Complete the following steps to deploy the Android and iOS versions of the mobile client to the Application Center (which is acting as the company's application store):

1. From the Worklight server, start Application Center using this address:
`http://localhost:9080/applicationcenter`
2. Log in with the appcenteradmin user ID and the password that is defined in the Worklight Server `server.xml` file (see in , "Worklight Server `server.xml` file" on page 323).
3. On the Applications tab, click **Add Application**.
4. Click in the File text field. In the File Upload dialog that opens, locate the directory that contains the Android application package. Select the package file and click **Open**. The file is uploaded to Worklight Server and a confirmation note is displayed, as shown in Figure 4-18.

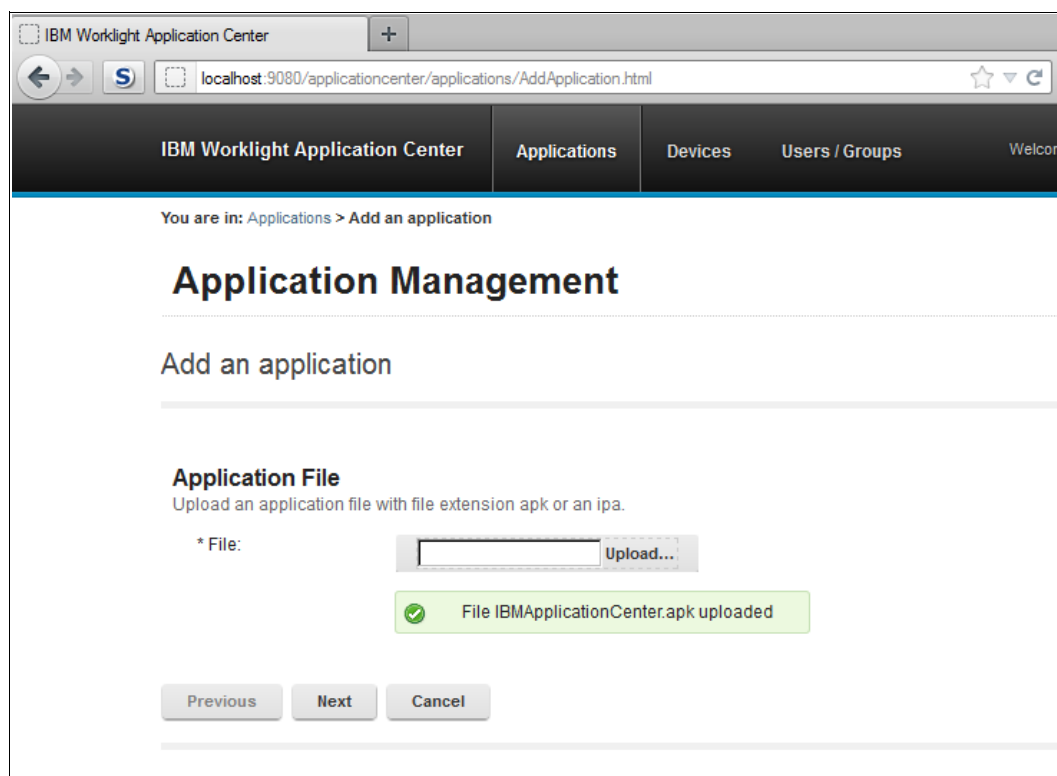


Figure 4-18 Confirmation of Android mobile client upload

5. Click **Next** to display the Application Details page and select **Installer** check box to indicate that this application is an installer, as shown in Figure 4-19. This property indicates that the application is used to install other applications on a mobile device and send feedback on an application from the mobile device to Application Center. Usually, only the Application Center mobile client is defined as an installer.

The screenshot shows a web browser window with the URL `localhost:9080/applicationcenter/applications/AddApplication.html`. The page title is "Application Management" and the sub-header is "Add an application". Below this, the "Application Details" section contains the following fields:

- Package:** `com.ibm.appcenter` (Identifies the application)
- Internal Version:** `1` (Internal version number used to compare versions)
- Commercial Version:** `1.0` (Version displayed on the mobile device)
- Label:** `IBM App Center` (Label of the application as defined by the developer)
- Author:** `appcenteradmin` (User who has uploaded this application)
- Description:** `Application Center Mobile Client for Android` (2048 characters maximum)
- Recommended:** ☐ (This application will be listed as a recommended application on the mobile device)
- Installer:** ☒ (Indicates whether this application is an installer) - This field is highlighted with a red box.
- Active:** ☒ (An active application can be installed on a device)
- Ready for production:** ☒ (Indicates whether this application is ready for production)

At the bottom of the form are three buttons: "Previous", "Finish", and "Cancel".

Figure 4-19 Declaring that the mobile client is an Installer

6. Click **Finish**.
7. Repeat steps 4 on page 85, through 6 for the iOS version of the mobile client. After the clients are deployed, they are listed in Application Center, as shown in Figure 4-20.

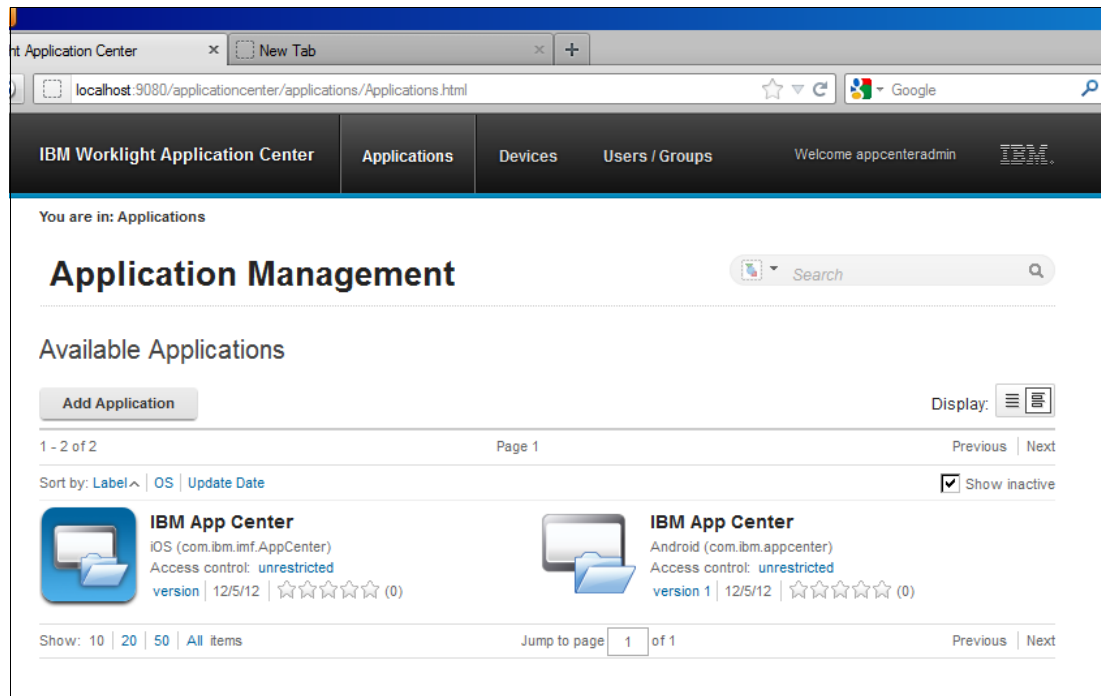


Figure 4-20 Application Center mobile clients shown in IBM Worklight Application Center

4.6 Introducing Worklight Studio projects

In IBM Worklight Studio, work is organized in projects. Within a single project, you can develop applications for various environments, including the following environments:

- ▶ Mobile environments, such as iPhone, Blackberry, and Android
- ▶ Desktop environments, such as Windows 8, Windows 7, and Mac OS
- ▶ Web environments, such as Facebook and iGoogle

All developer assets, including source code, libraries, and resources, are placed in a project folder. Within the project folder, each application you create has a main application folder and several sub-folders:

- ▶ A common folder to store code that is shared between all environments, such as HTML, CSS, and JavaScript code
- ▶ Additional folders to store code that is specific to each supported environment, such as Java code for Android or Objective-C code for iOS
- ▶ An adapters folder, to store the code of the adapters that the application requires to collect data from back-end systems

There will be several Worklight Studio projects at Airline Company A, where developers will fill many roles in the creation of the mobile luggage tracking application and the related adapters.

4.7 Creating the adapters

Adapters are used to connect the mobile application to the company's back-end business services and to implement the server-side authentication model.

One HTTP adapter is created to allow the authentication model to validate user access to the application. Another HTTP adapter is created to establish a connection to the back-end business services. The second adapter retrieves luggage information from the LTIS using XML-based RESTful services, and translates the data retrieved from this system into JavaScript Object Notation (JSON) format, which is used by the mobile application.

The following subsections detail the steps required to create and test the adapters using IBM Worklight Studio and then export the adapters to IBM Worklight Server.

4.7.1 Setting up the adapter project

Both the authentication adapter and the business services adapter will be developed within the same Worklight Studio project. To create this project, use the following steps:

1. Start IBM Worklight Studio with a workspace that you will use for this book.

Each time you start Worklight Studio, ensure that you are using the same workspace. You can switch workspaces in Worklight Studio by selecting **File** → **Switch Workspace** and then selecting an existing workspace or selecting **Other** and then creating a new workspace by using a new workspace name in the Workspace Launcher dialog.

2. Create a new project by selecting **File** → **New** → **Worklight Project**.

3. Enter `AirlineAdapterProject` for the Project name. Select **Hybrid Application** from the Project Templates list, as shown in Figure 4-21, and click **Next**.

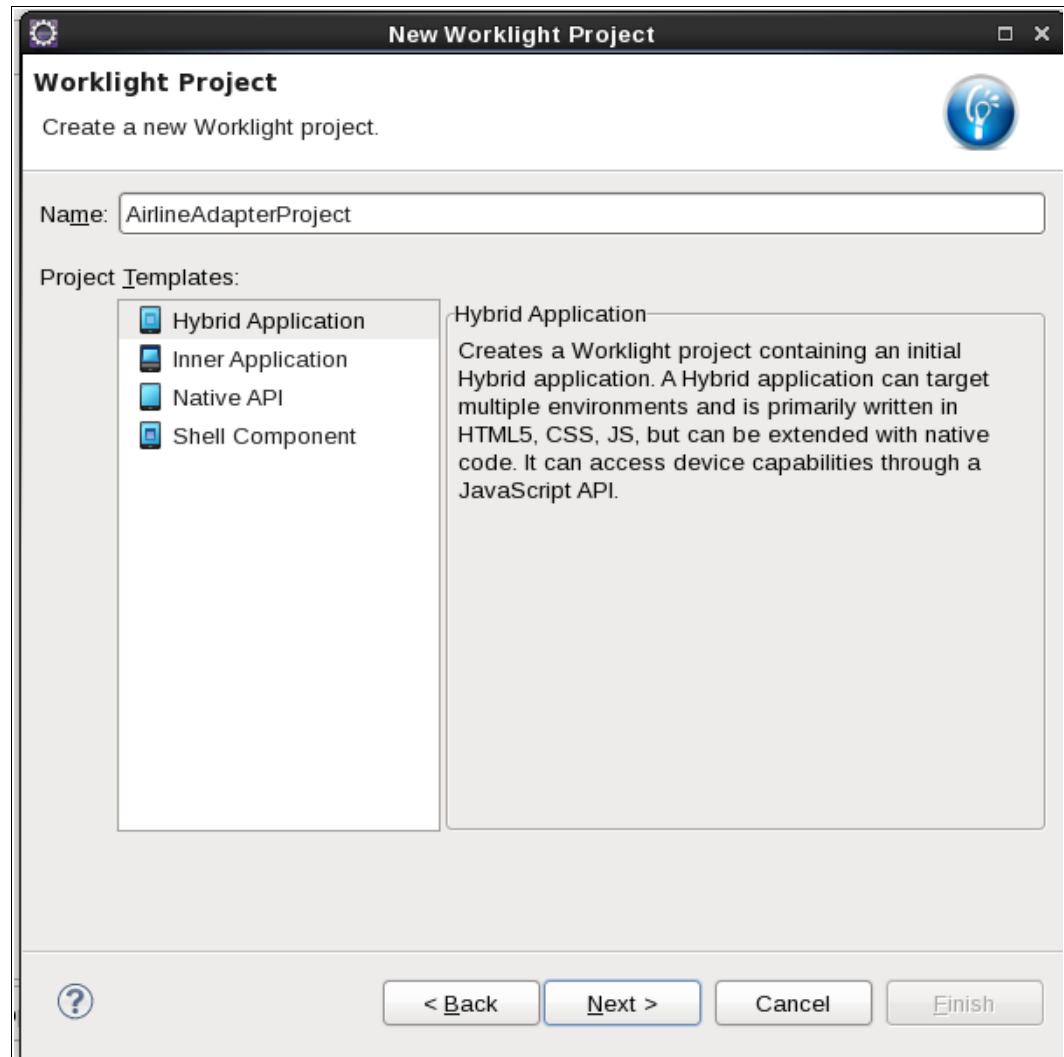


Figure 4-21 Creating the adapter project

4. Enter LuggageTracker for the Application name, as shown in Figure 4-22, and then click **Finish**.

The screenshot shows the 'New Worklight Project' dialog box with the 'Hybrid Application' tab selected. The 'Application name' field is filled with 'LuggageTracker'. Below it, the 'jQuery Mobile Installation' section has an unchecked checkbox for 'Add jQuery Mobile' and a 'Location' field. A note states: 'The Worklight runtime provides jQuery Core. Select the jQuery Mobile resources you would like to import:'. The 'Sencha Touch Installation' section has an unchecked checkbox for 'Add Sencha Touch' and a 'Location' field with a 'Folder...' button. The 'Dojo installation' section has an unchecked checkbox for 'Add Dojo Toolkit' and a note: 'Dojo toolkit support will be added to the application.' At the bottom, there are buttons for '< Back', 'Next >', 'Cancel', and 'Finish' (which is highlighted).

Figure 4-22 Entering the application name

5. If you are prompted to open the Design perspective, click **No**. You might not be prompted if you have saved a preference on what perspective to open when you created a previous Worklight project.

After clicking Finish in the New Worklight Project dialog, the new project is displayed in Project Explorer and the Application Descriptor is displayed in the Application Descriptor Editor. The Application Descriptor file stores the metadata for the Worklight application, such as the display name and description.

4.7.2 Creating the authentication adapter

As described in 4.4, “Planning for security” on page 75, Airline Company A is implementing adapter-based authentication for LuggageTracker. The adapter is responsible for receiving and validating the user credentials from the mobile application. This allows the underlying security mechanisms to be changed without impacting the mobile application itself.

There are three basic stages to implementing the authentication adapter:

- ▶ Creating the adapter
- ▶ Configuring the authentication realm
- ▶ Testing the adapter

Creating the adapter

To create the HTTP authentication adapter, use the following steps:

1. In Project Explorer, right-click the LuggageTracker folder (within the apps folder) and create a new adapter by selecting **New** → **Worklight Adapter**, as shown in Figure 4-23.

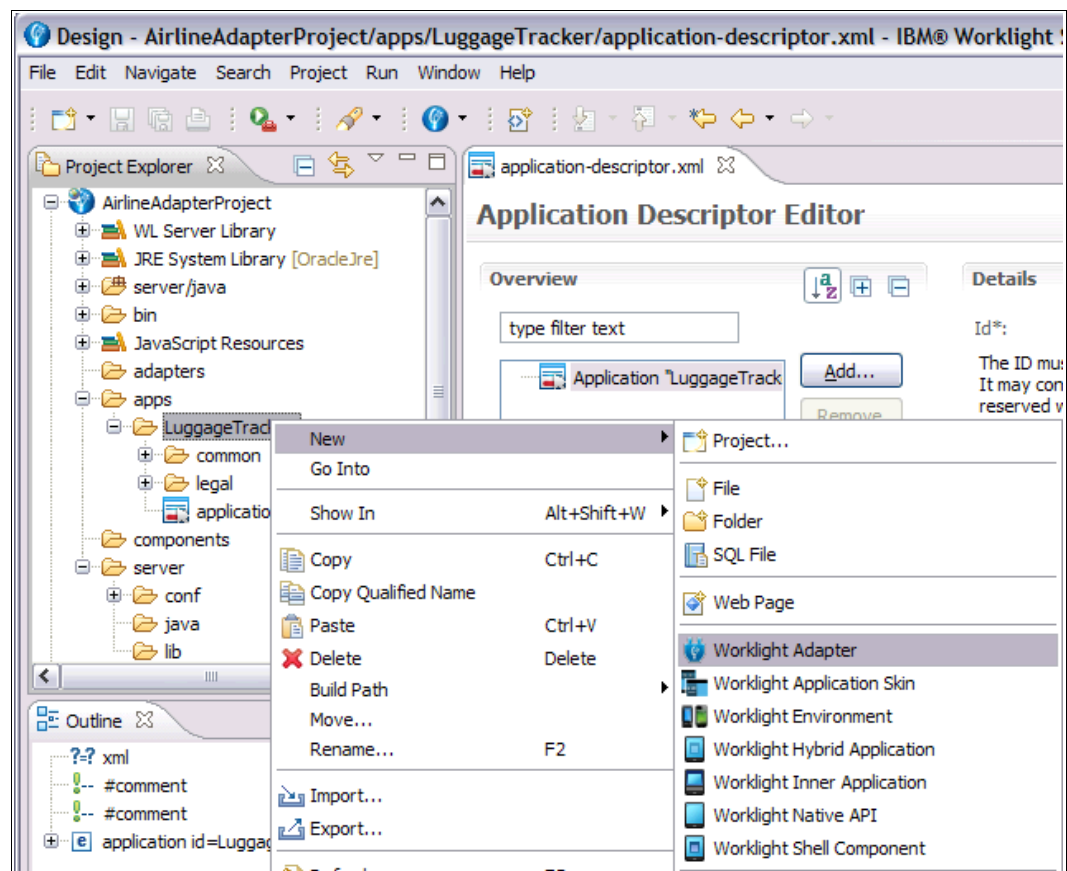


Figure 4-23 Creating a new Worklight Adapter

2. In the New Worklight Adapter dialog, select **HTTP Adapter** from the Adapter type list, and enter LuggageTrackerAdapter for the Adapter name, as shown in Figure 4-24.

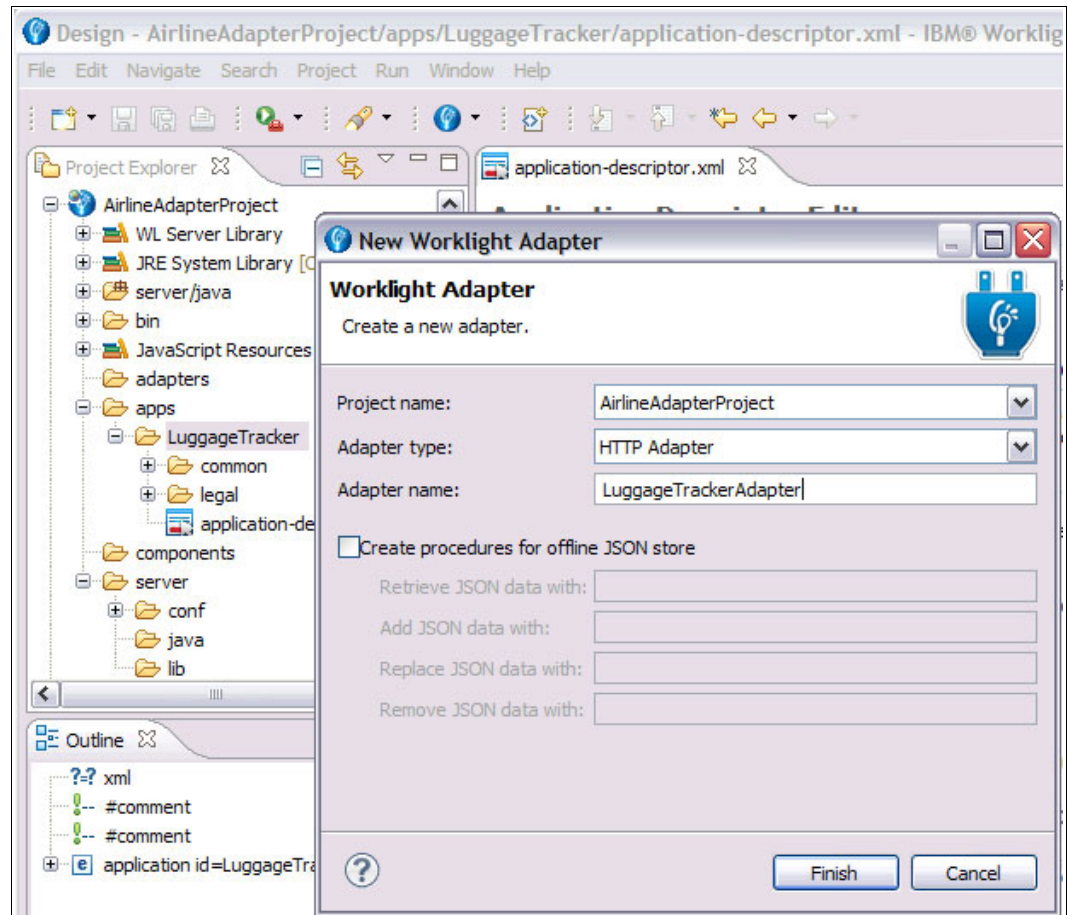


Figure 4-24 Setting the adapter name and type

3. Click **Finish** to create the adapter.

The New Worklight Adapter dialog is closed and the LuggageTrackerAdapter project now has a new adapters folder, and, within it, a new LuggageTrackerAdapter folder containing several auto-generated files, as shown in Figure 4-25.

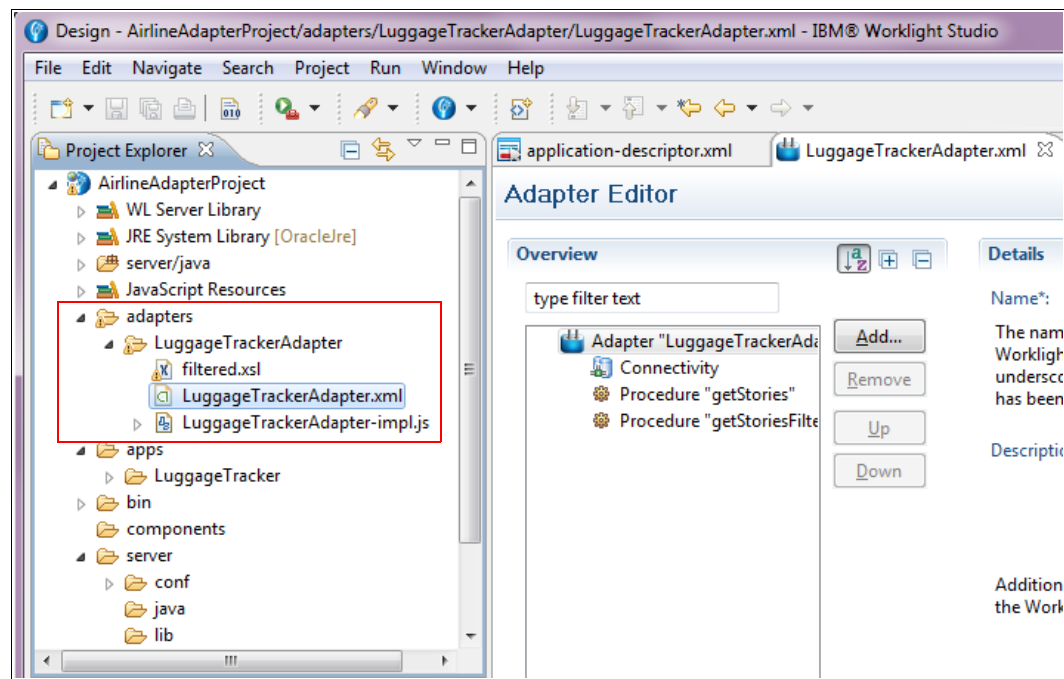


Figure 4-25 New auto-generated adapter files

The auto-generated files are used in developing the adapter, as follows:

- The LuggageTrackerAdapter.xml file contains the metadata for the adapters, including the name, description connection parameters, and procedure details.
 - The LuggageTrackerAdapter-impl.js file contains the JavaScript code for the procedures listed in the LuggageTrackerAdapter.xml file.
 - The filtered.xsl file contains the Extensible Stylesheet Language (XSL) transformation definition that can be used to filter the data returned from the back-end services.
4. After clicking Finish in the New Worklight Adapter dialog, the LuggageTrackerAdapter.xml file is automatically opened in the Adapter Editor. If this does not occur, open the file by right-clicking the file and selecting **Open With** → **Adapter Editor**.
 5. Switch to the **Source** tab in the Adapter Editor to show the XML source.
 6. In the XML source, find the <wl:adapter> tag block that was automatically created when the file was generated and replace it with the XML shown in Example 4-8. When doing this, replace *server_name* and *server_port* with the server URL and port of your enterprise authentication provider.

Example 4-8 Replacement adapter tag block in LuggageTrackerAdapter.xml

```
<wl:adapter name="LuggageTrackerAdapter"
  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>LuggageTrackerAdapter</displayName>
  <description>Provides access to authentication model</description>
  <connectivity>
```

```

        <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
            <protocol>http</protocol>
            <domain>server_name</domain>
            <port>server_port</port>
        </connectionPolicy>
        <loadConstraints maxConcurrentConnectionsPerNode="2" />
    </connectivity>
    <procedure name="submitAuthentication"/>
    <procedure name="onLogout"/>
</wl:adapter>

```

7. Save and close the LuggageTrackerAdapter.xml file.
8. Open the LuggageTrackerAdapter-impl.js file and replace the auto-generated JavaScript with the three authentication functions shown in Example 4-9. The submitAuthentication and onLogout functions implement the procedures defined in the adapter XML shown in the previous example and the onAuthRequired function is used by the submitAuthentication function.

Example 4-9 Adding the authentication functions

```

function submitAuthentication(username, password){

    //The lines below implement the simulated User Registry
    //Either use the sample user registry provided with this book, or
    //change these lines to utilize your own user registry.
    var reg = com.ibm.itso.saw210.security.UserRegistryFactory.getRegistry();
    var authenticationResult = reg.authenticate(username, password);

    if (authenticationResult){
        var userIdentity = {
            userId: username,
            displayName: username,
            attributes: {
                userType: "CSR"
            }
        };
        WL.Server.setActiveUser("LuggageTrackerRealm", userIdentity);
        return {
            isLoginSuccessful:true,
            authRequired: false
        };
    }
    else {
        WL.Server.setActiveUser("LuggageTrackerRealm", null);
        return onAuthRequired(null, "Invalid login credentials");
    }
}

function onAuthRequired(headers, errorMessage){
    WL.Logger.debug("onAuthRequired");
    errorMessage = errorMessage ? errorMessage : null;

    return {
        isLoginSuccessful: false,
        authRequired: true,
        errorMessage: errorMessage
    }
}

```



```

    };
}

function onLogout(){
    WL.Server.setActiveUser("LuggageTrackerRealm", null);
    WL.Logger.debug("Logged out");
}

```

9. If you will use your own user registry, change the two lines shown in bold to use your user registry to authenticate the user credentials.

If you do not already have a user registry and want to use the simple file-based user registry used in this book, the source code is provided in “User registry” on page 339. You must compile the Java classes and package the compiled classes in a Java Archive (JAR) file; add them to the *AirlineAdapterProject* in the *server/lib* directory.

10. Save and close the *LuggageTrackerAdapter-impl.js* file.

Configuring the authentication realm

As described in 4.4, “Planning for security” on page 75, the server-side component of the airline’s new luggage tracking application consists of an authentication realm, named *LuggageTrackerRealm*, which defines the authentication process. To add the authentication realm to the Worklight Server configuration, use the following steps:

1. Edit the *authenticationConfig.xml* file, which is located in the *server/conf* directory of the adapter project, and define the *LuggageTrackerRealm* by adding the XML shown in Example 4-10 inside the existing *<realms>* tag.

Example 4-10 LuggageTrackerRealm definition

```

<realm name="LuggageTrackerRealm" loginModule="LuggageTrackerLoginModule">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function"
        value="LuggageTrackerAdapter.onAuthRequired" />
    <parameter name="logout-function" value="LuggageTrackerAdapter.onLogout" />
</realm>

```

The *<realm>* tag defines the name of the realm, in this case *LuggageTrackerRealm*, and the name of the login module, *LuggageTrackerLoginModule*.

The designated class name, *com.worklight.integration.auth.AdapterAuthenticator*, indicates that the server-side part of the authenticator is defined in the adapter. This type of adapter-based authentication requires two functions: a login function and a logout function. The login-function parameter is automatically invoked by the Worklight authentication framework when an attempt is made to access a protected resource. For the airline scenario, the login function has the following name:

LuggageTrackerAdapter.onAuthRequired

The logout-function parameter is invoked automatically when an attempt to log out is detected and has the following name in this scenario:

LuggageTrackerAdapter.onLogout

2. Add a login module to the <loginModules> section of authenticationConfig.xml, as shown in Example 4-11, inside the existing <loginModules> tag.

Example 4-11 Login module

```
<loginModule name="LuggageTrackerLoginModule">
  <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
</loginModule>
```

In this example, the NonValidatingLoginModule class name means that no additional validation is performed by the Worklight platform.

3. Save and close the authenticationConfig.xml file.

Testing the adapter

To test an adapter, it must first be deployed in the development environment. To deploy and test the authentication adapter that you just created, use the following steps:

1. Right-click the adapter name, LuggageTrackerAdapter, and select **Run As** → **Deploy Worklight Adapter**, as shown in Figure 4-26.

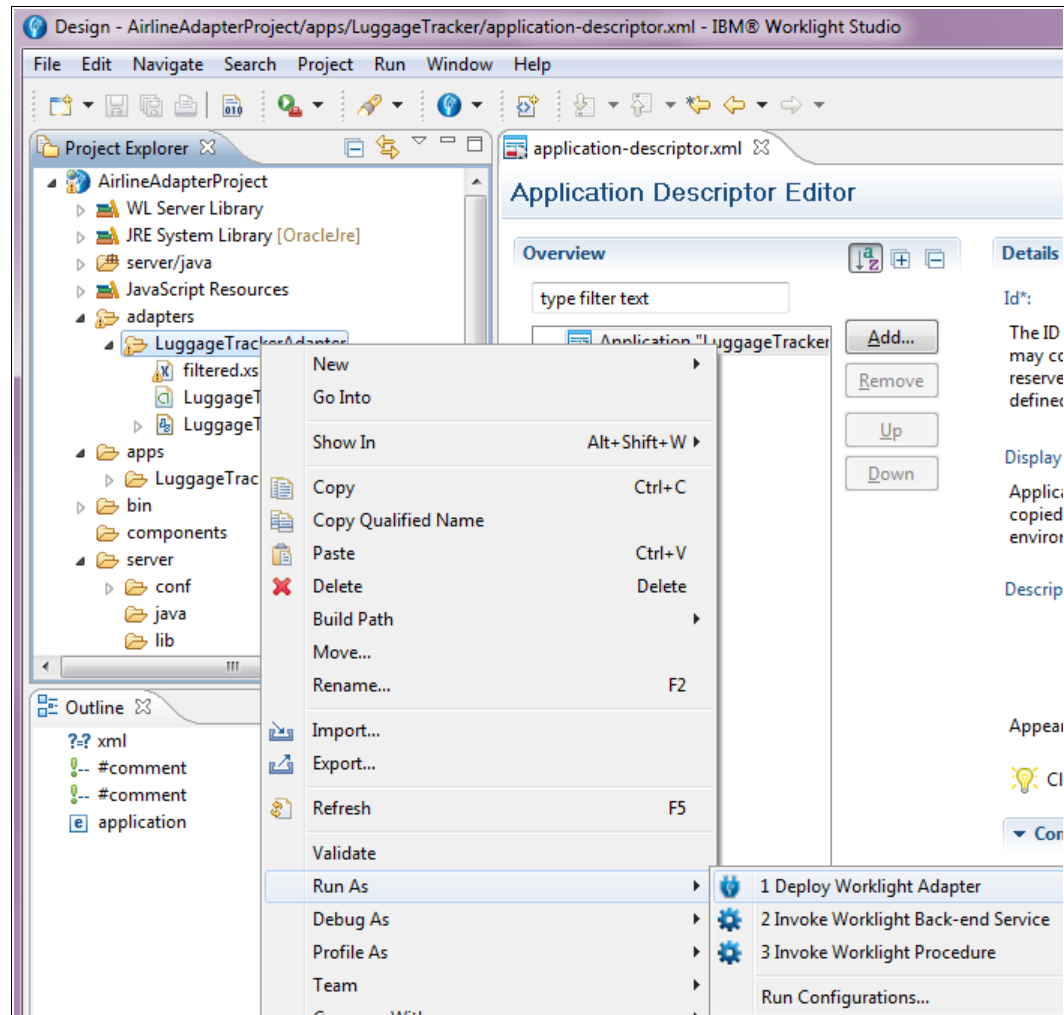


Figure 4-26 Deploying the authentication adapter

2. Monitor the Worklight Console, shown in Figure 4-27, where a message will appear when the adapter has been deployed successfully. If the console is not visible, click the Console icon in the lower right corner of Worklight Studio.

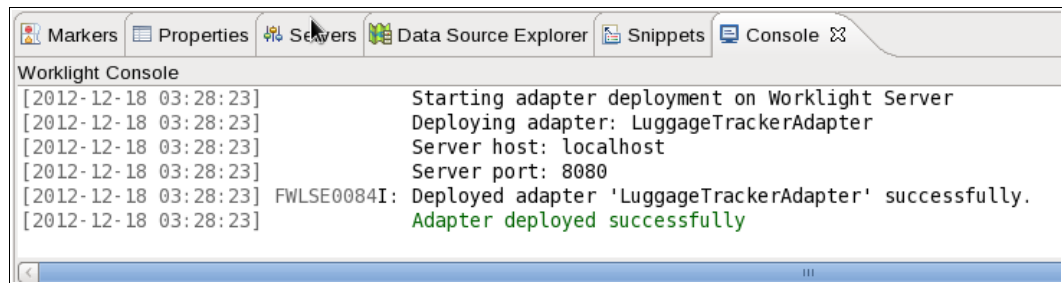


Figure 4-27 Message indicating successful deployment of the adapter

3. With the adapter successfully deployed, you can now test it. Right-click the adapter name, `LuggageTrackerAdapter`, and select **Run As** → **Invoke Worklight Procedure** to display the Edit Configuration window. The names of the project and adapter are already filled in.
4. Select **submitAuthentication** from the Procedure name list to indicate that you want to run the adapter's `submitAuthentication` procedure.
5. This procedure needs the credentials that validate a user's access of the mobile application. Enter a valid user name and password in the Parameters field (in this case, both the user name and password were set to `worklight` in one of the user ID/password combinations in the `users.properties` file that was defined in "User registry" on page 339). The Edit Configuration window now looks like it does in Figure 4-28 on page 98.

Hint: be sure to include the double-quotation marks around the user ID and password as shown in the figure.

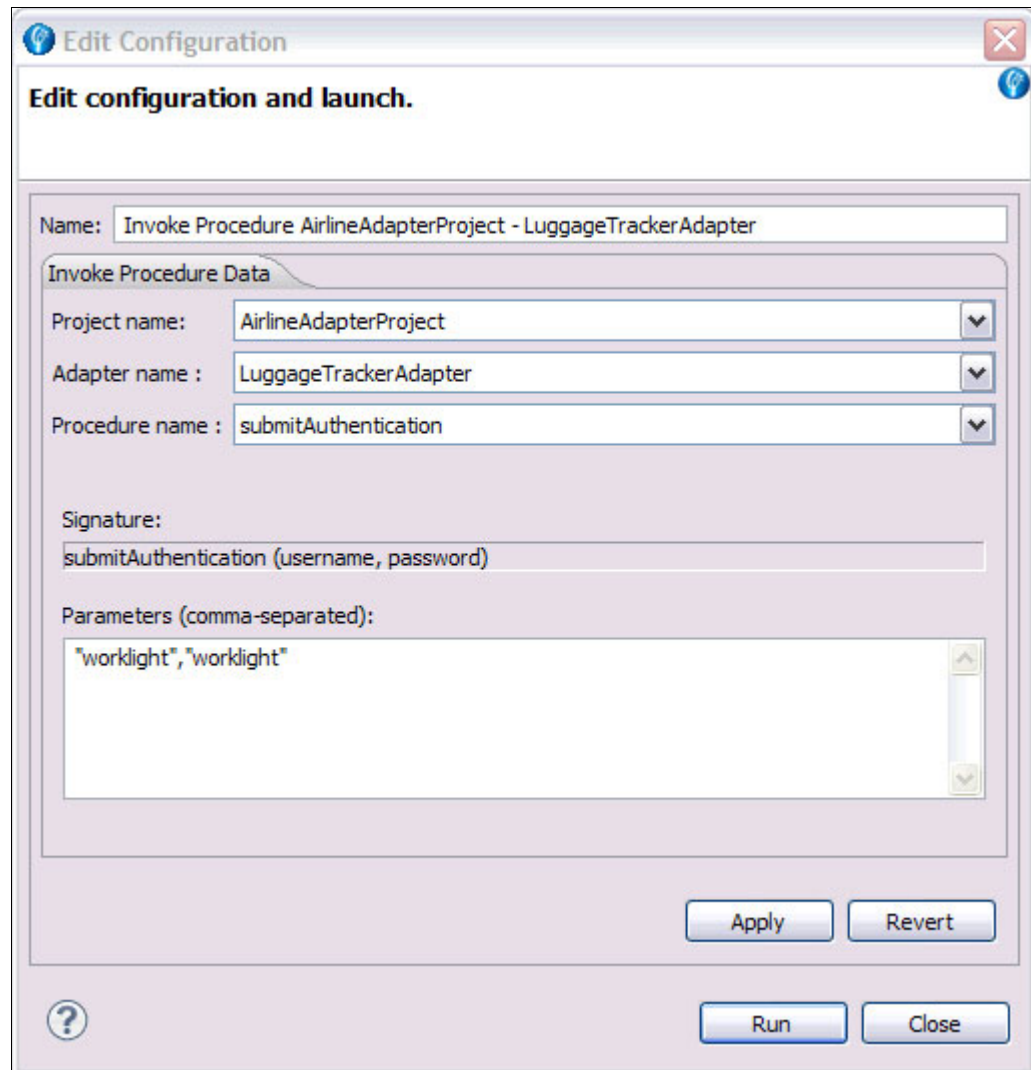


Figure 4-28 Specifying the parameters to test the authentication adapter

6. Click **Apply** and then **Run** to save the configuration and run the authentication adapter.
7. Switch to the Invoke Procedure Result view to see the results of the operation using JSON notation. The submission of valid credentials results in a successful procedure invocation, as shown in Figure 4-29.

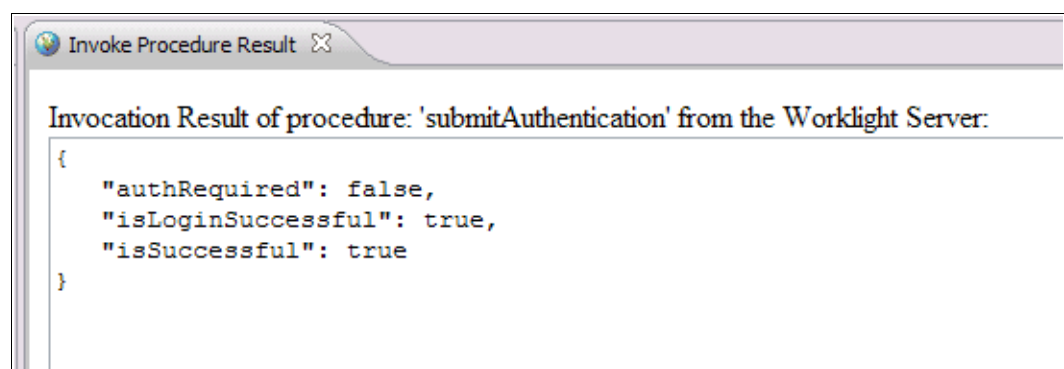


Figure 4-29 Notice of successful authentication

8. To complete testing the adapter, repeat steps 3 on page 97 through 7 on page 98 but enter an invalid user name and password in the **Parameters** field. When you click **Apply** and then **Run**, the submission of the invalid credentials results in an unsuccessful procedure invocation; the results tab will be similar to Figure 4-30.

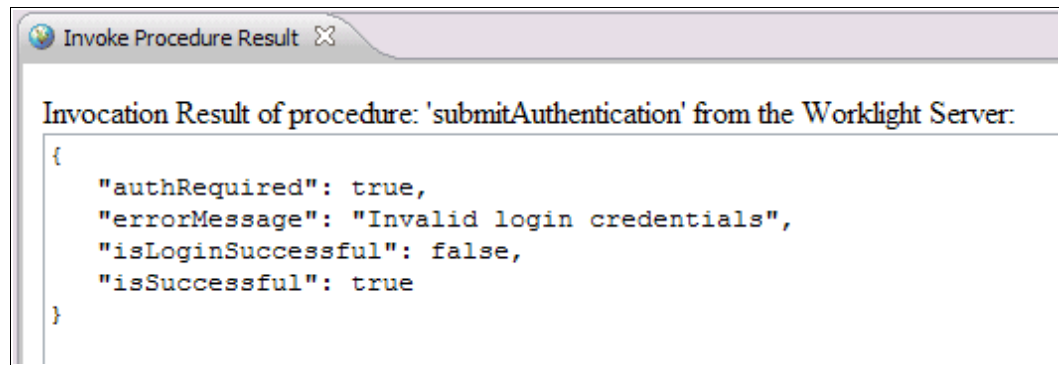


Figure 4-30 Notice of unsuccessful authentication

4.7.3 Creating the business services adapter

As described in 4.3, “Providing enterprise services” on page 71, Airline Company A is using a Worklight adapter to provide access to the existing back-end services. These back-end services will be used to retrieve data from the Luggage Tracking Information System (LTIS) and update the passenger’s delivery address, if necessary.

There are three stages to implementing the business services adapter:

- ▶ Creating the adapter
- ▶ Adding filtering capabilities
- ▶ Testing the adapter

Creating the adapter

The business services HTTP adapter is part of the AirlineAdapterProject that also contains the authentication adapter that was just created. To create the business services adapter, use the following steps:

1. In Project Explorer, right-click the adapters folder and select **New** → **Worklight Adapter**.
2. In the New Worklight Adapter dialog, select **HTTP Adapter** from the Adapter type list, and enter BusinessServicesAdapter for the Adapter name.
3. Click **Finish** to create the adapter. A new folder named BusinessServicesAdapter is created (see Figure 4-31 on page 100). Within it are several auto-generated files similar to the files that were generated when the authentication adapter was created.

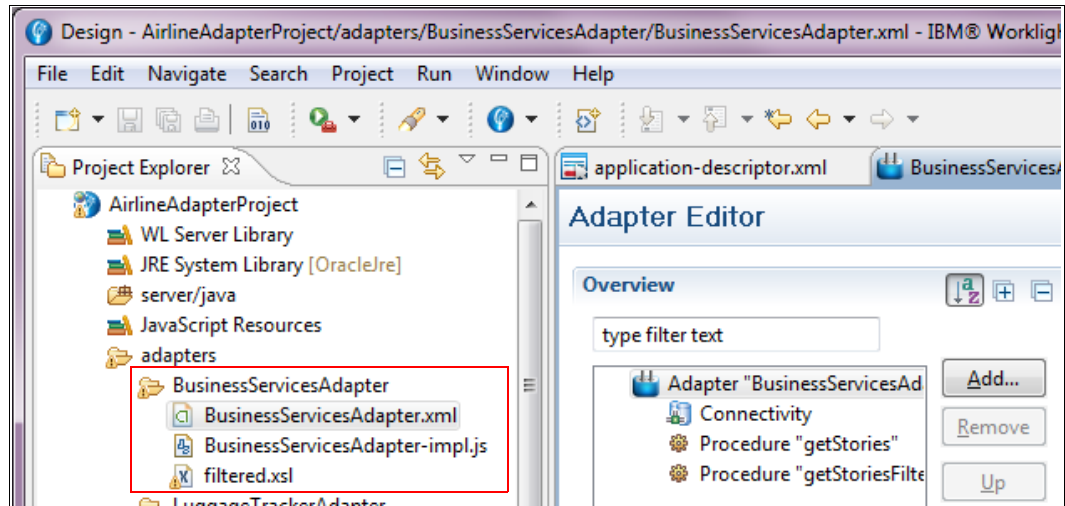


Figure 4-31 New BusinessServicesAdapter

4. In the New Worklight Adapter dialog, the BusinessServicesAdapter.xml file is automatically opened in the Adapter Editor. If this does not occur, open the file by right-clicking the file and selecting **Open With** → **Adapter Editor**.
5. Switch to the **Source** tab if it is not shown.
6. In the XML source, find the <wl:adapter> tag block that was automatically created when the file was auto-generated; replace it with the XML shown in Example 4-12. When doing this, replace *server_name* and *server_port* with the server URL and port of your enterprise business services provider. In this scenario, the business services are simulated using a web application deployed to a separate WebSphere Liberty profile server.

Example 4-12 Replacement adapter tag block in BusinessServicesAdapter.xml

```
<wl:adapter name="BusinessServicesAdapter"
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>BusinessServicesAdapter</displayName>
  <description>Provides access to enterprise business services</description>
  <connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>http</protocol>
      <domain>server_name</domain>
      <port>server_port</port>
    </connectionPolicy>
    <loadConstraints maxConcurrentConnectionsPerNode="2" />
  </connectivity>

  <procedure name="getLuggageInformation" />
  <procedure name="addLuggageDeliveryInformation" />

</wl:adapter>
```

7. Save and close the BusinessServicesAdapter.xml file.

8. Open the `BusinessServicesAdapter-impl.js` file and replace the auto-generated JavaScript with the JavaScript shown in Example 4-13, which contains the procedures that were defined in the `BusinessServicesAdapter.xml` file (Example 4-12 on page 100).

Example 4-13 Adding the business services functions

```
function getLuggageInformation(luggageId) {
    var input = {
        method : 'get',
        returnerContentType : 'xml',
        path : 'AirlineREST/jaxrs/luggage/'+luggageId
    };

    return WL.Server.invokeHttp(input);
}

function addLuggageDeliveryInformation(trackingId, name, addressLine1,
addressLine2, city, state, zipCode, phoneNumber, comments) {
    var input = {
        method : 'post',
        path : 'AirlineREST/jaxrs/luggage/',
        parameters : {
            'trackingId' : trackingId,
            'name' : name,
            'addressLine1' : addressLine1,
            'addressLine2' : addressLine2,
            'city' : city,
            'state' : state,
            'zipCode' : zipCode,
            'phoneNumber' : phoneNumber,
            'comments' : comments
        }
    };

    return WL.Server.invokeHttp(input);
}
```

The `getLuggageInformation` function retrieves back-end services data using the HTTP GET method and expects XML data in the response. It sets the invocation path to the service base URL plus the `luggageId` parameter.

The `addLuggageDeliveryInformation` function sends data, including parameters such as the passenger's delivery address, using an HTTP POST request to the same base URL.

9. Save and close the `BusinessServicesAdapter-impl.js` file.

Adding filtering capabilities

The business services adapter's transformation feature is used to filter out unnecessary data that is returned from the back-end services and to map attribute names. The feature applies a stylesheet transformation, using an XSL file, to the incoming message and converts it to the data structure used by the application.

To enable message filtering and transformation at the adapter lever, use the following steps:

1. Open the business services adapter's `filtered.xsl` file, which was auto-generated when the adapter was created.
2. Replace all of the file's contents with the stylesheet definition shown in Example 4-14. The stylesheet is used by the adapter transformation function to map the original XML luggage object from the back-end LTIS to a new luggage object that contains only the fields needed by the mobile application. The transformation function is also where any name transformations can be done to de-couple the adapter from the back-end services in case of a service definition change.

Example 4-14 Replacement XML for adapter transformations in filtered.xsl file

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:h="http://www.w3.org/1999/xhtml">
  <xsl:output method="text"/>
  <xsl:template match="/luggage">
    {
      'luggage': {
        'trackingId' : '<xsl:value-of select="trackingId"/>',
        'ownerName' : '<xsl:value-of select="ownerName"/>',
        'status' : '<xsl:value-of select="status"/>',
        'flightNumber' : '<xsl:value-of select="flightNumber"/>',
        'currentLocation' : '<xsl:value-of select="currentLocation"/>',
        'nextLocation' : '<xsl:value-of select="nextLocation"/>',
        'finalDestination' : '<xsl:value-of select="finalDestination"/>',
        'comments' : '<xsl:value-of select="comments"/>',
        'lastUpdate' : '<xsl:value-of select="lastUpdate"/>',
        'deliveryAddress' : {
          'addressLine1' : '<xsl:value-of
            select="deliveryAddress/addressLine1"/>',
          'addressLine2' : '<xsl:value-of
            select="deliveryAddress/addressLine2"/>',
          'city' : '<xsl:value-of
            select="deliveryAddress/city"/>',
          'comments' : '<xsl:value-of
            select="deliveryAddress/comments"/>',
          'name' : '<xsl:value-of select="deliveryAddress/name"/>',
          'phoneNumber' : '<xsl:value-of
            select="deliveryAddress/phoneNumber"/>',
          'state' : '<xsl:value-of
            select="deliveryAddress/state"/>',
          'zipCode' : '<xsl:value-of
            select="deliveryAddress/zipCode"/>'
        }
      }
    }
  </xsl:template>
</xsl:stylesheet>
```

3. Save and close the `filtered.xsl` file.

4. Add the filtering by opening the `BusinessServicesAdapter-impl.js` file and replacing the `getLuggageInformation` function with the new method shown in Example 4-15.

Example 4-15 Updating BusinessServicesAdapter-impl.js to add filtering

```
function getLuggageInformation(luggageId) {  
    var input = {  
        method : 'get',  
        returnerContentType : 'xml',  
        path : 'AirlineREST/jaxrs/luggage/'+luggageId,  
        transformation : {  
            type : 'xslFile',  
            xslFile : 'filtered.xsl'  
        }  
    };  
  
    return WL.Server.invokeHttp(input);  
}
```

The transformation element that was added to the existing procedure definition (see highlighted portion of the example) includes information about the transformation process. It defines that the transformation is done using an XSL file named `filtered.xsl`, which is the file you updated in Example 4-14 on page 102.

5. Save and close the `BusinessServicesAdapter-impl.js` file.

Testing the adapter

The new business services adapter is deployed and tested through the same process used for the authentication adapter. In your IBM Worklight Studio development environment, use the following steps:

1. Right-click the adapter name, `BusinessServicesAdapter`, and select **Run As → Deploy Worklight Adapter**.
2. Monitor the Worklight Console, where a message will appear to confirm that the adapter was deployed successfully. If the console is not visible, click the Console icon in the lower right corner of Worklight Studio.
3. With the adapter successfully deployed, you can now test it. Right-click the adapter name, `BusinessServicesAdapter`, and select **Run As → Invoke Worklight Procedure** to display the Edit Configuration window. The Project name and Adapter name are already filled in.
4. Select **getLuggageInformation** from the Procedure name list and enter a valid luggage identifier number into the Parameters field.
5. Click **Apply** and then **Run** to save the configuration and run the business services adapter.
6. Switch to the Invoke Procedure Result view to see the results of the operation using JSON notation, as shown in Figure 4-32 on page 104.

```
{
  "errors": [
  ],
  "info": [
  ],
  "isSuccessful": true,
  "luggage": {
    "comments": "some comments",
    "currentLocation": "a location",
    "deliveryAddress": {
      "addressLine1": "some address information",
      "addressLine2": "some address information",
      "city": "a city",
      "comments": "some comments",
      "name": "a name",
      "phoneNumber": "a phone number",
      "state": "a state",
      "zipCode": "a zip code"
    },
    "finalDestination": "a destination",
    "flightNumber": "a flight number",
    "lastUpdate": "timestamp",
    "nextLocation": "a location",
    "ownerName": "a name",
    "status": "the status",
    "trackingId": "1"
  },
  "responseHeaders": {
    "Content-Language": "en-US",
    "Content-Length": "648",
    "Content-Type": "application/xml",
    "Date": "Thu, 13 Dec 2012 16:05:08 GMT",
    "Server": "WebSphere Application Server",
    "X-Powered-By": "Servlet/3.0"
  },
  "statusCode": 200,
  "statusReason": "OK",
  "warnings": [
  ]
}
```

Figure 4-32 Notice of a successful luggage information retrieval

7. Repeat steps 3 on page 103 through 6 on page 103 with an invalid luggage identifier number to see the results of an unsuccessful invocation.

4.8 Creating the mobile application

The LuggageTracker mobile application is created by a mobile application developer using IBM Worklight Studio. With Worklight Studio, the developer can build the application's user interface in a Web 2.0 UI style using Dojo toolkit and jQuery. Airline Company A is using jQuery to build the user interface for LuggageTracker. Although the IBM jQuery tools are part of IBM Worklight Studio and are installed with the product, the jQuery Mobile library is not included and must be downloaded from the jQuery Mobile website:

<http://jquerymobile.com/>

Download and extract the file (for example, jquery.mobile-1.2.0.zip for the 1.2.0 version of jQuery Mobile). You use this file in 4.8.1, "Creating the user interface" on page 105.

4.8.1 Creating the user interface

You are now ready to build the user interface elements using the graphical editor in IBM Worklight Studio. The LuggageTracker application consists of four screens, each of which contains specific elements:

- ▶ Login
 - Text input field for Username
 - Text input field for Password
 - Login button
- ▶ Luggage ID input
 - Text input field for Luggage ID
 - Scan Bar Code button
 - Retrieve Information button
 - Logout button
- ▶ Luggage information
 - Text input field for Current Status
 - Text input field for Owner Name
 - Text input field for Current Location
 - Text input field for Next Destination
 - Text input field for Final Destination
 - Text input field for Last update
 - Text area field for Comments
 - Change Delivery Address button
 - Close button
- ▶ Luggage delivery address
 - Text input field for Luggage ID
 - Text input field for Name
 - Text input field for Address line 1
 - Text input field for Address line 2
 - Text input field for City
 - Text input field for State
 - Text input field for Zip Code
 - Text input field for Phone Number
 - Text area field for Comments
 - Save and return button
 - Cancel button

For purposes of this book, detailed instructions are provided for creating only the two most complicated user interface screens, Luggage ID input and Luggage information. Abbreviated instructions for creating the Login and Luggage delivery address screens will be supplied at the conclusion of this subsection,

Creating the user interface project

The user interface is created using a separate IBM Worklight Studio project from the one (AirlineAdapterProject) that was created in 4.7, “Creating the adapters” on page 88. To create the Worklight project for the LuggageTracker application, use the following steps:

1. Start IBM Worklight Studio, if it is not already started.
2. Select **File** → **New** → **Worklight Project** to display the New Worklight Project wizard, shown in Figure 4-33.

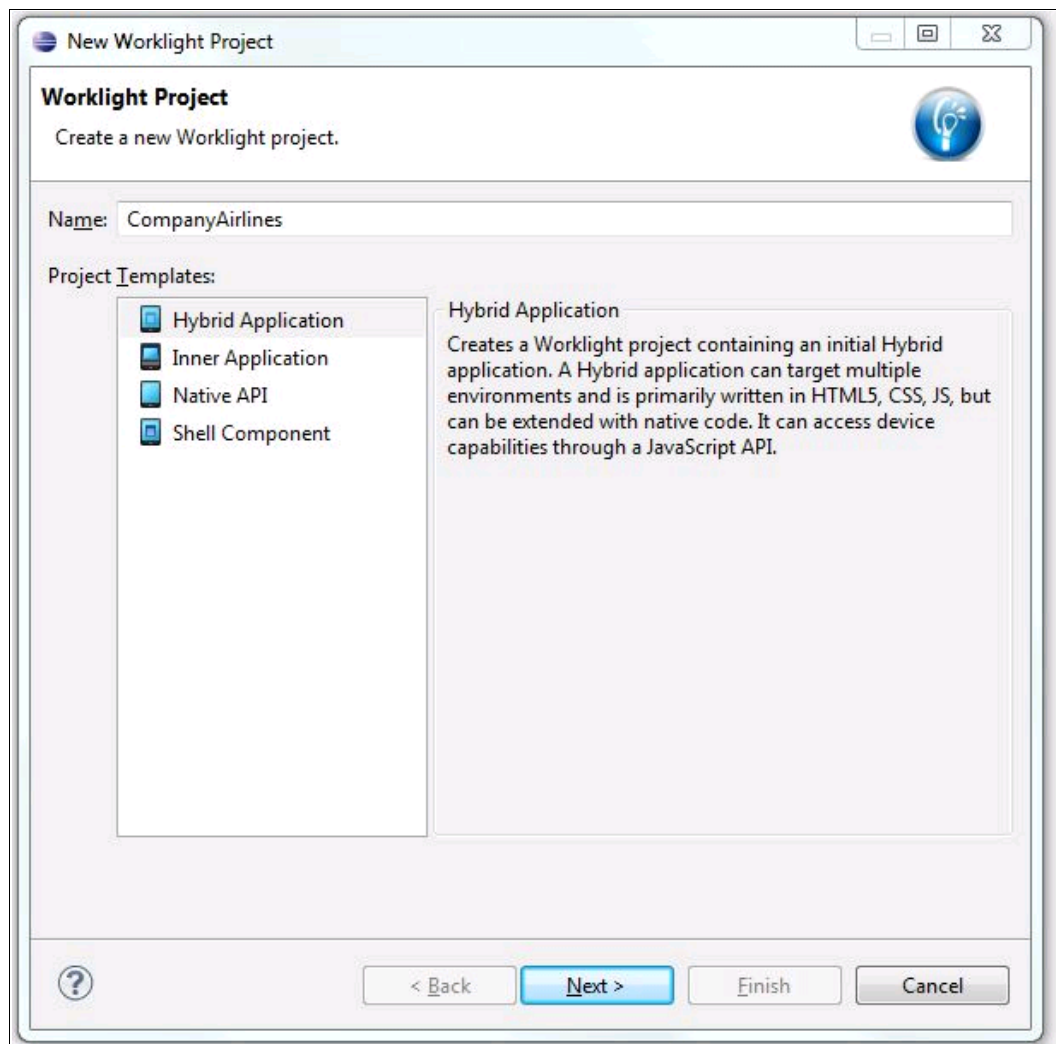


Figure 4-33 Creating a new project with the New Worklight Project wizard

3. Enter the project name, `CompanyAirLines`, and select a template for the project. IBM Worklight Studio includes four project templates, depending on the type of application that you will be creating:
 - Native API
Use this for applications that will be built entirely using native programming with Worklight APIs. This type of project is used when you are targeting a specific mobile platform.
 - Shell Component
Use this to write native code components that can be reused later. This shell provides a wrapper around an inner application (a web application) and provides the integration from that inner (web) application to the native device APIs.
 - Inner Application
Use this to create a web application that reuses a shell component application and can contain HTML, JavaScript, and CSS parts. In this project type, the shell component must be created in advance of creating the inner application.
 - Hybrid Application
Use this to create a mobile application with both web and native shell components, but does not require the shell component to be created before you begin.

Because the `LuggageTracker` application includes both the user interface and a shell component for the bar code scanner, select **Hybrid Application** from the list of Project Templates, and then click **Next**.

4. In the next wizard window, shown in Figure 4-34, configure the hybrid application. In the Application name field, enter the name of the mobile application, LuggageTracker.

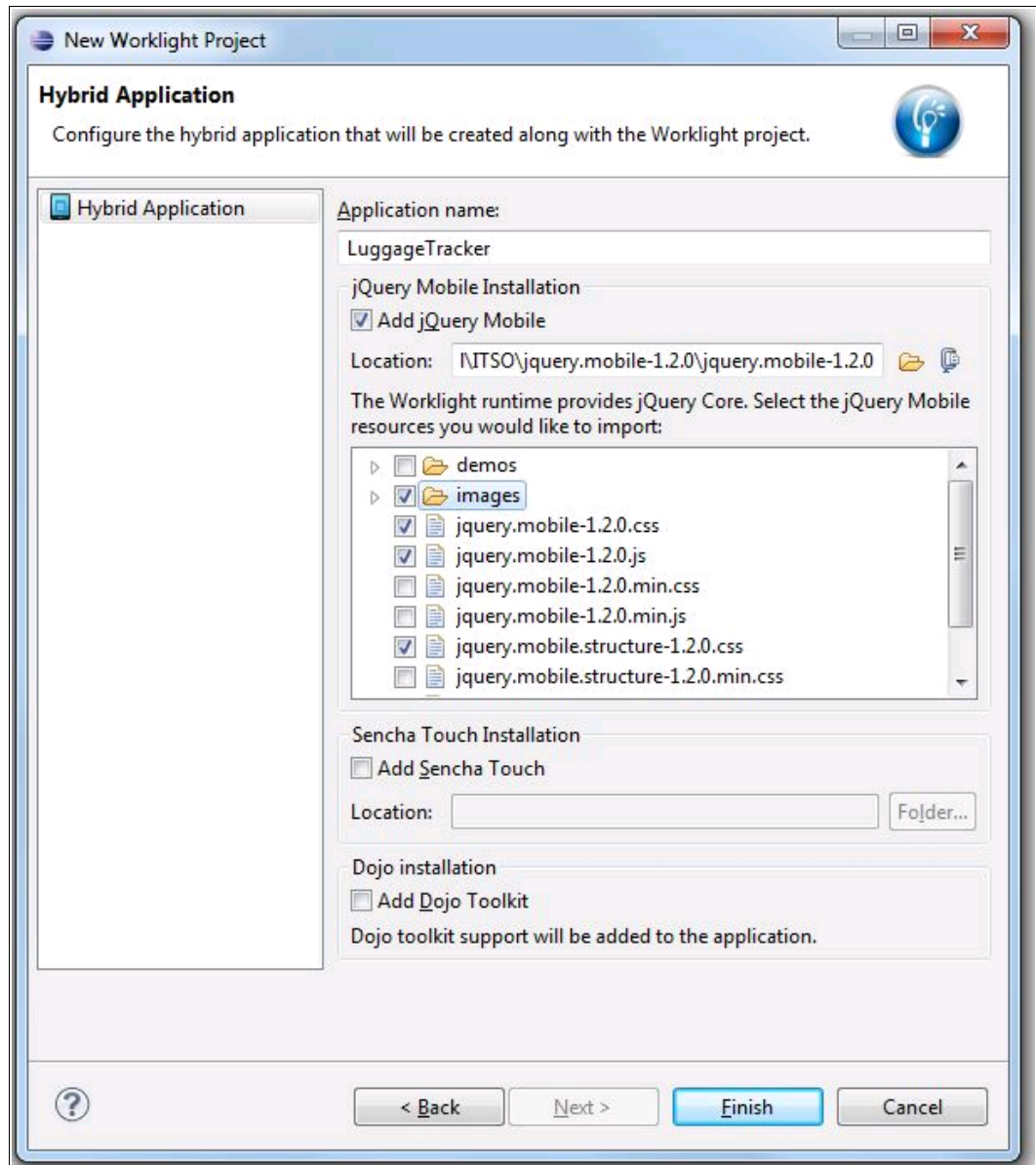


Figure 4-34 Configuring the hybrid application properties for LuggageTracker

5. The user interface will be built using jQuery Mobile, so select the **Add jQuery Mobile** check box to add the jQuery Mobile library.
6. Immediately beneath this check box, click the folder icon next to the Location field and select the location where you extracted the jQuery Mobile library in 4.8, “Creating the mobile application” on page 104.
7. In the list of resources, select the images folder and the three files that are shown in Figure 4-34, and then click **Finish**.

The project is now created and the Application Descriptor Editor, shown in Figure 4-35, is opens automatically. The mobile application developer can now close the application descriptor file and editor and move on to developing the user interface screens.

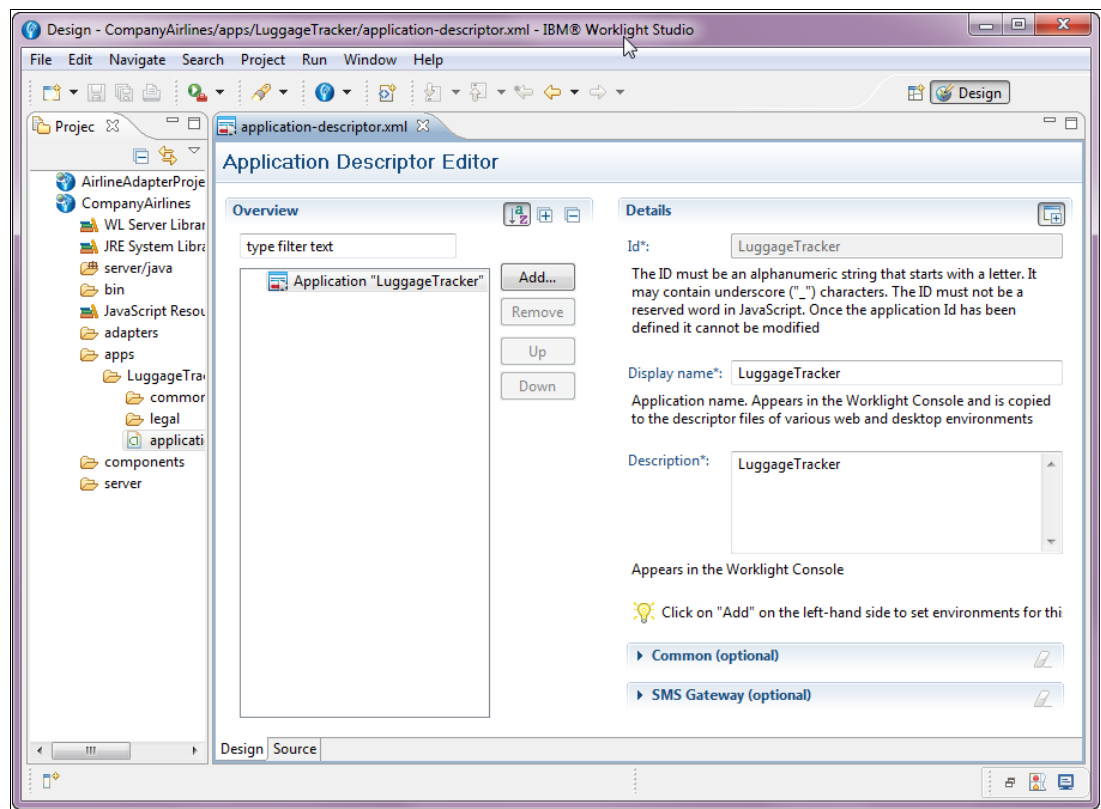


Figure 4-35 LuggageTracker application descriptor shown in the Descriptor Editor

Building the user interface screens

To create the user interface screens, you edit the mobile application main file, `LuggageTracker.html`, which was automatically generated when the LuggageTracker application was created (in this case, as part of the creation of the CompanyAirlines project).

Complete the following steps:

1. In Project Explorer, expand the CompanyAirlines project until the files in the LuggageTracker folder are visible.
2. Open the `LuggageTracker.html` file, which displays in the Rich Page Editor portion of the Design perspective, as shown in Figure 4-36 on page 110.

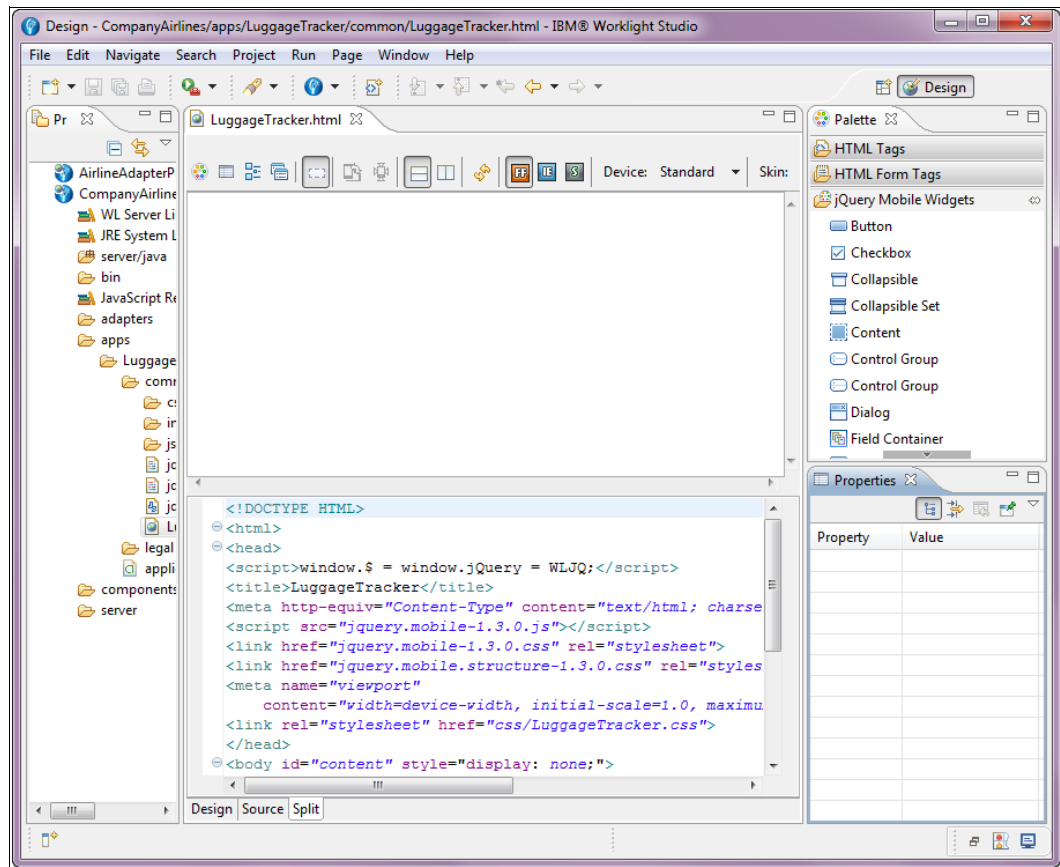


Figure 4-36 LuggageTracker.html shown in the Rich Page Editor

The Design perspective has multiple views and is customizable. Use the following main views in creation of the LuggageTracker application:

- The Project Explorer, where you expanded your project, shows all of your projects and the files and libraries within each project.
- The Palette shows the list of available user interface components or widgets that you can drag and drop with the Rich Page Editor.
- The Rich Page Editor contains your LuggageTracker.html file contents and is the editor for the mobile application user interface. At the top is the Design section, where you can build your user interface in a graphical manner by dragging and dropping widgets from the Palette. The bottom section, labeled Source, shows the actual source code associated with what is shown in the Design section. Edits to this source code are reflected automatically in the Design section.
- The Properties view is where you can edit a component's attributes and event values.

Common header and footer

All four screens of the mobile application are defined in the `LuggageTracker.html` file, and all have the same header and footer, as shown in Figure 4-37. The header contains the heading text, Luggage Tracker; the footer is a solid black bar that is shown only for cosmetic purposes.

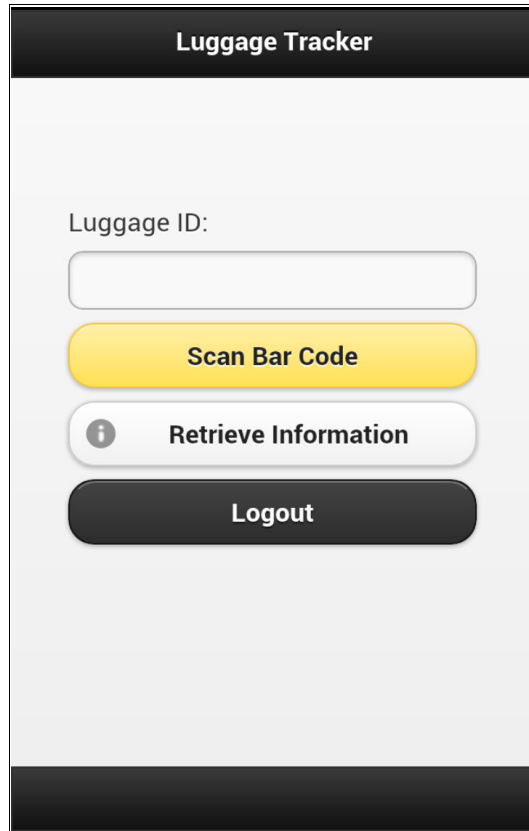


Figure 4-37 Luggage ID input screen showing header and footer

Still working in the `LuggageTracker.html` file, add the header and footer as follows:

1. A jQuery Header widget is responsible for the screen header. Find the Header widget in the jQuery Mobile Widgets section of the Palette and drag it into the Add widgets here portion of the Design section (see Figure 4-38 on page 112).
2. Change the header text from Header to Luggage Tracker, either by editing the source code in the Source section or by double-clicking the header widget, changing the text, and then clicking a space outside of the widget.
3. To add the footer, find the Footer widget in the jQuery Mobile Widgets section of the Palette and drag it into the Add widgets here portion of the Design section.

4. Edit the Footer widget, as described in step 2 on page 111, to remove the default text so that the footer is empty. The results are shown in Figure 4-38.

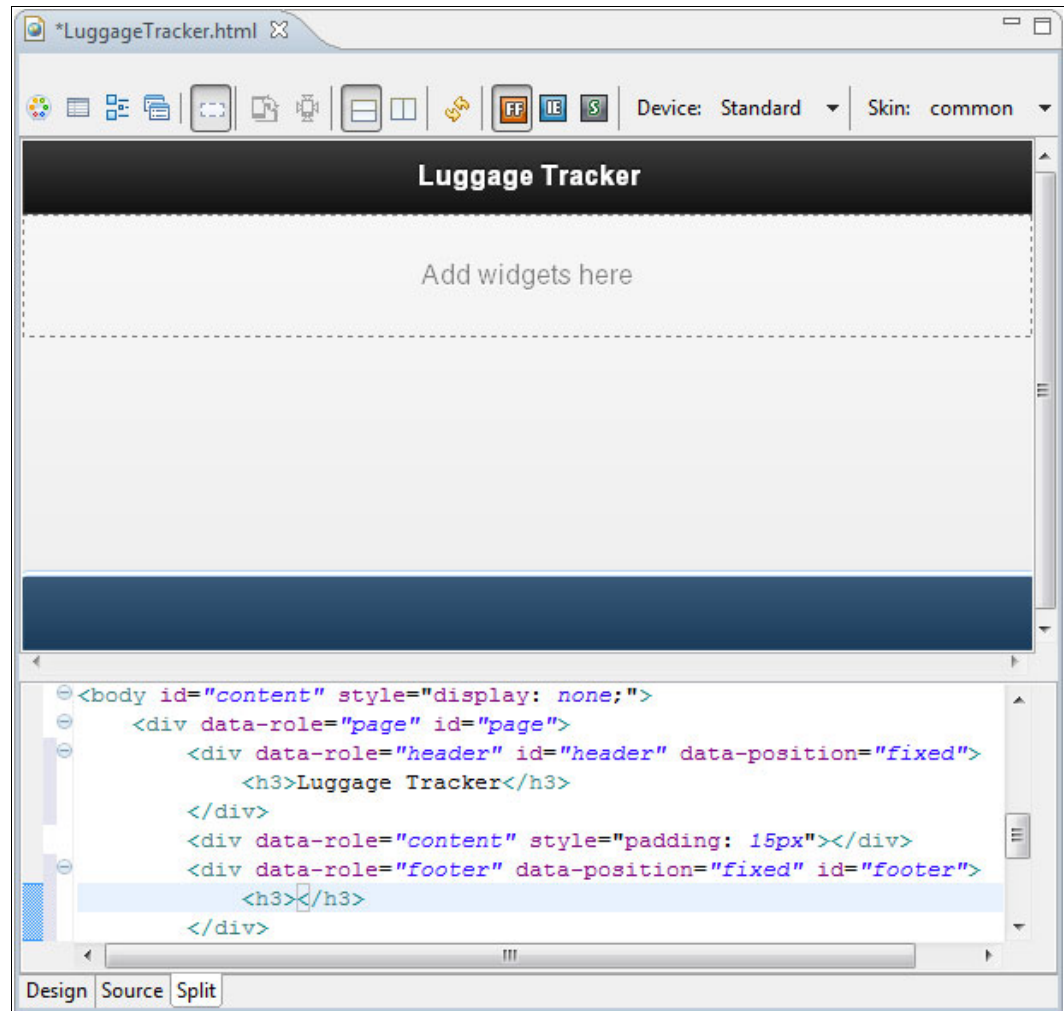


Figure 4-38 Header and footer created in the Rich Page Editor

Luggage ID input screen

Each screen of the application is defined within a series of screen-specific Content widgets that are placed between the header and footer widgets in the Rich Page Editor. The application hides and shows these Content widgets to display particular screens to the user.

The Luggage ID input screen, shown in Figure 4-37 on page 111, consists of one input field widget and three button widgets, all contained within a Content widget. Still working in the `LuggageTracker.html` file, create the luggage ID input screen, following these steps:

1. Find the Content widget in the jQuery Mobile Widgets Palette and drag it into the Add widgets here section. This widget will hold additional widgets related to the input field and buttons. The dashed lines in the Design view indicate the user interface layout and padding between the various elements within it.
2. Select the Content widget and in the Properties view, change the widget ID attribute to `luggageIdInput`. The widget ID gives the screen a meaningful internal name so that the mobile application developer can more easily reference the widget later, for example to hide or show the screen. If you use a different ID for the widget, you must make the same change in the JavaScript functions that you add later.

3. Drag a Text Input widget to inside the Content widget. A Text Input widget has three attributes that must be changed from their default values:
 - Label: Double-click the label text on the Design tab, which puts the label into edit mode and then change the default text to Luggage ID:. You can also edit the HTML directly on the Source tab and change the label text between the <label> and </label> tags.
 - Name: Click the input field on the Design tab and in the Properties view, change the Name property on the Tag tab to LuggageTrackingID. You can also edit the HTML directly on the Source tab and change the name attribute of the <input> tag to LuggageTrackingID.
 - ID: Click the input field on the Design tab and in the Properties view, change the ID property on the Tag tab to LuggageTrackingID. You can also edit the HTML directly on the Source tab and change the ID attribute of the <input> tag to LuggageTrackingID.

If you use a different ID for the Text Input widget, you will have to make the same change in the JavaScript functions that you will add later.
4. Drag a Button widget into the Content widget and drop it beneath the Text Input widget. A button widget has two attributes that must be changed from their default values:
 - Text: Double-click the button on the Design tab, which puts the button text into edit-mode and then change the default text to Scan Bar Code. You can also edit the HTML directly on the Source tab and change the button text between the <a> and tags.
 - ID: Click the button on the Design tab and in the Properties view, change the ID property on the Tag tab to ScanBarcodeButton. You can also edit the HTML directly on the Source tab and change the ID attribute of the <input> tag to ScanBarcodeButton.
5. To add some look-and-feel effects, you can change the button color using a built-in theme, and give the button rounded corners. From the jQuery tab of the Properties view, select theme **e** and **rounded corners**. The changes are immediately shown in the Design section.
6. Drag another Button widget into the Content widget and drop it beneath the previous Button widget. Using the methods described in step 4, change the text of the button to Retrieve Information and change the id to retrieveInfoButton.
7. Add additional look-and-feel effects, including an icon, as you did in step 5. From the jQuery tab of the Properties view, select **rounded corners**, and from Icon menu, choose the **Info** icon. The changes are shown in the Design section.
8. Drag a third Button widget into the Content widget and drop it beneath the previous Button widget. Using the methods described in step 4, change the text of the button to Logout and change the id to logoutButton.
9. Add look-and-feel effects as you did in steps 5 and 7. From the jQuery tab of the Properties view, select theme **a** and **rounded corners**. The changes are shown in the Design section.

The screenshot shows the Sencha Touch IDE interface. The main window displays a mobile application titled "Luggage Tracker". The app has a black header bar with the title. Below the header, there is a dashed box containing the main content area. The content area includes a text input field labeled "Luggage ID:", a yellow button labeled "Scan Bar Code", a grey button labeled "Retrieve Information" with an information icon, and a black button labeled "Logout". The right sidebar shows the "Palette" with various HTML and jQuery Mobile widgets. The bottom pane shows the HTML source code for the application, including the LuggageTracker header and the content area with the form elements.

The Scan Bar Code button calls a bar code scanner plug-in that is packaged with the luggage tracking application and uses the mobile device's camera to scan the bar code on a passenger's luggage receipt and then display the luggage ID number in the Luggage ID field. When the Retrieve Information button is clicked, the luggage details are retrieved from the airline's back-end systems and displayed in the Luggage information screen, as shown in Figure 4-40 on page 115.

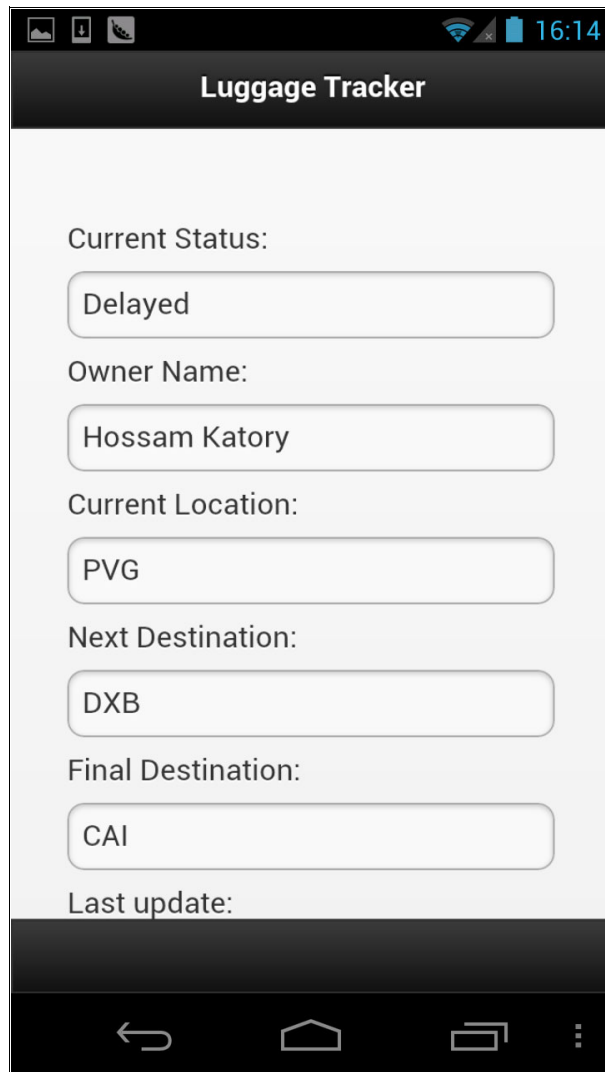


Figure 4-40 Luggage information screen

Luggage information screen

The Luggage information screen has multiple text fields to display information about the passenger's bag, a larger text area for comments to be entered, and two buttons. While continuing to work in the `LuggageTracker.html` file, create the Luggage information screen:

1. Find the Content widget in the jQuery Mobile Widgets Palette and drag it to the section named Add widgets here.
2. Select the Content widget and in the Properties view, change the widget ID attribute from `pageContent0` to `l_Information`. If you use a different ID for the widget, you will have to make the same change in the JavaScript functions that you will add later.
3. Drag the needed number of Text Input widgets into the new Content widget. Then use the methods described in step 3 on page 113 to change the label, name, and ID attributes for each widget to what is shown in Table 4-3 on page 116.

Table 4-3 Text Input widget label text and name and ID attributes

Label text	Name and ID attributes
Current Status:	I_Status
Owner Name:	I_Owner
Current Location:	I_location
Next Destination:	I_NextLocation
Final Destination:	I_FinalLocation
Last update:	I_lastUpdate

- In the Design section, click each Text Input widget and then switch to the Tag tab in the Properties view. On the Tag tab, select **Read-only** as the Initial state of each widget. These fields only display information and do not accept input.

With the luggage information text fields added, the top portion of the screen displayed in Rich Page Editor will look similar to Figure 4-41.

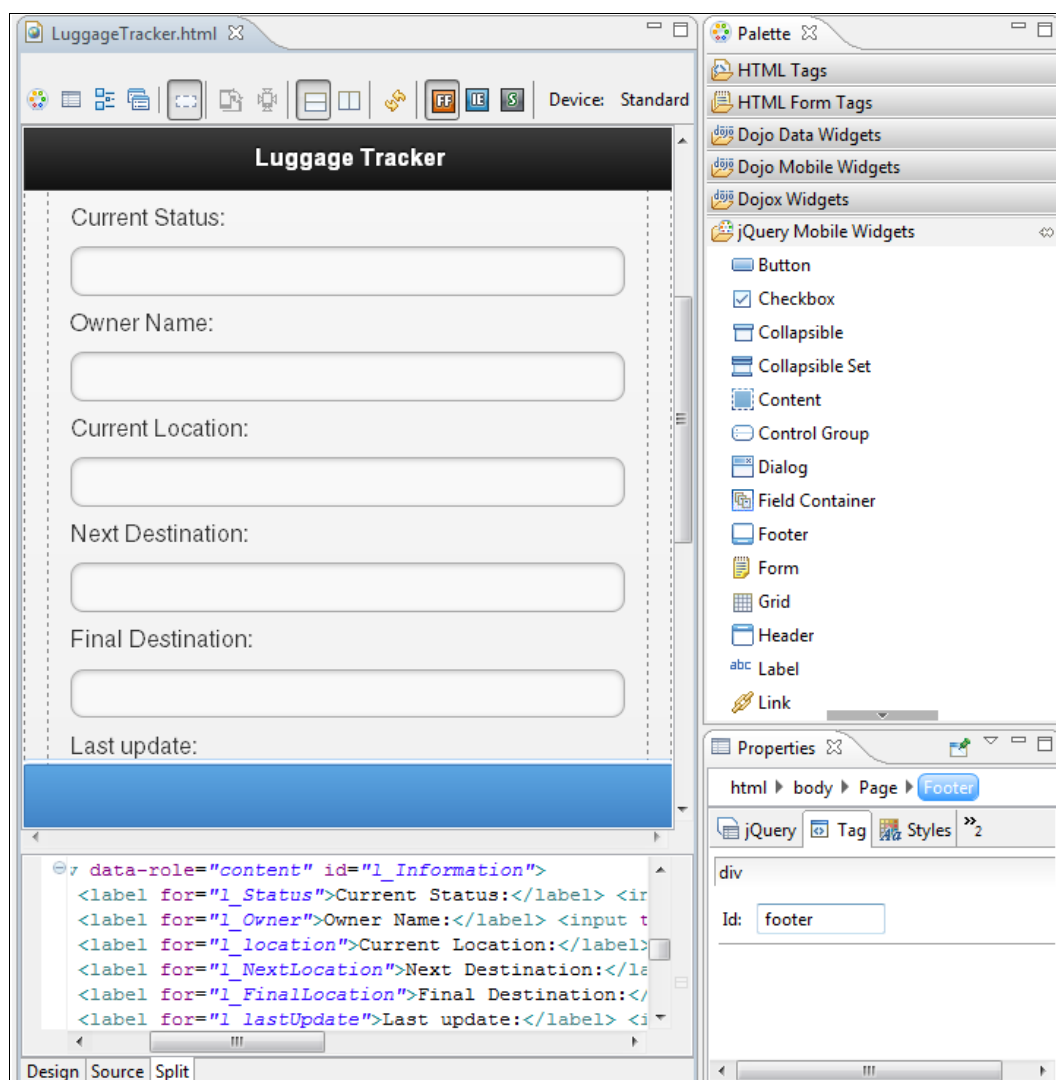


Figure 4-41 Top portion of Luggage information screen in Rich Page Editor

5. Next is the Text Area widget. However, because a Text Area widget does not include a label, you must first drag a Label widget into the Content widget (drop it beneath the last Text Input widget, labeled Last update) and then drag the Text Area widget to just beneath the Label widget. Then, make the following attribute changes:
 - Label: Double-click the label text on the Design tab, which puts the label into edit mode and then change the default text to `Comments :`. You can also edit the HTML directly on the Source tab and change the label text between the `<label>` and `</label>` tags.
 - Name: Click the text area on the Design tab and, in the Properties view, change the Name property on the Tag tab to `1_comments`. You can also edit the HTML directly on the Source tab and change the name attribute of the `<textarea>` tag to `1_comments`.
 - ID: Click the input field on the Design tab and, in the Properties view, change the ID property on the Tag tab to `1_comments`. You can also edit the HTML directly on the Source tab and change the ID attribute of the `<textarea>` tag to `1_comments`.
6. Drag a Button widget into the Content widget and drop it beneath the Text Area widget. Then, follow the method described in step 3 on page 113 to change the text of the button to `Change Delivery Address` and change the ID attribute to `changeAddressButton`.
7. Drag another Button widget into the Content widget and drop it beneath the Button widget you just added. Then, follow the method described in step 3 on page 113 to change the text of the button to `Close` and the ID attribute to `closeButton`.
8. To add look-and-feel effects, you can change the button color using a built-in theme, add an icon, and give the button rounded corners. From the jQuery tab of the Properties view, select theme **b** and **rounded corners**, and from Icon menu, choose the **Home** icon. The changes are immediately shown in the Design section.

When you finish, your Rich Page Editor display looks similar to Figure 4-42.

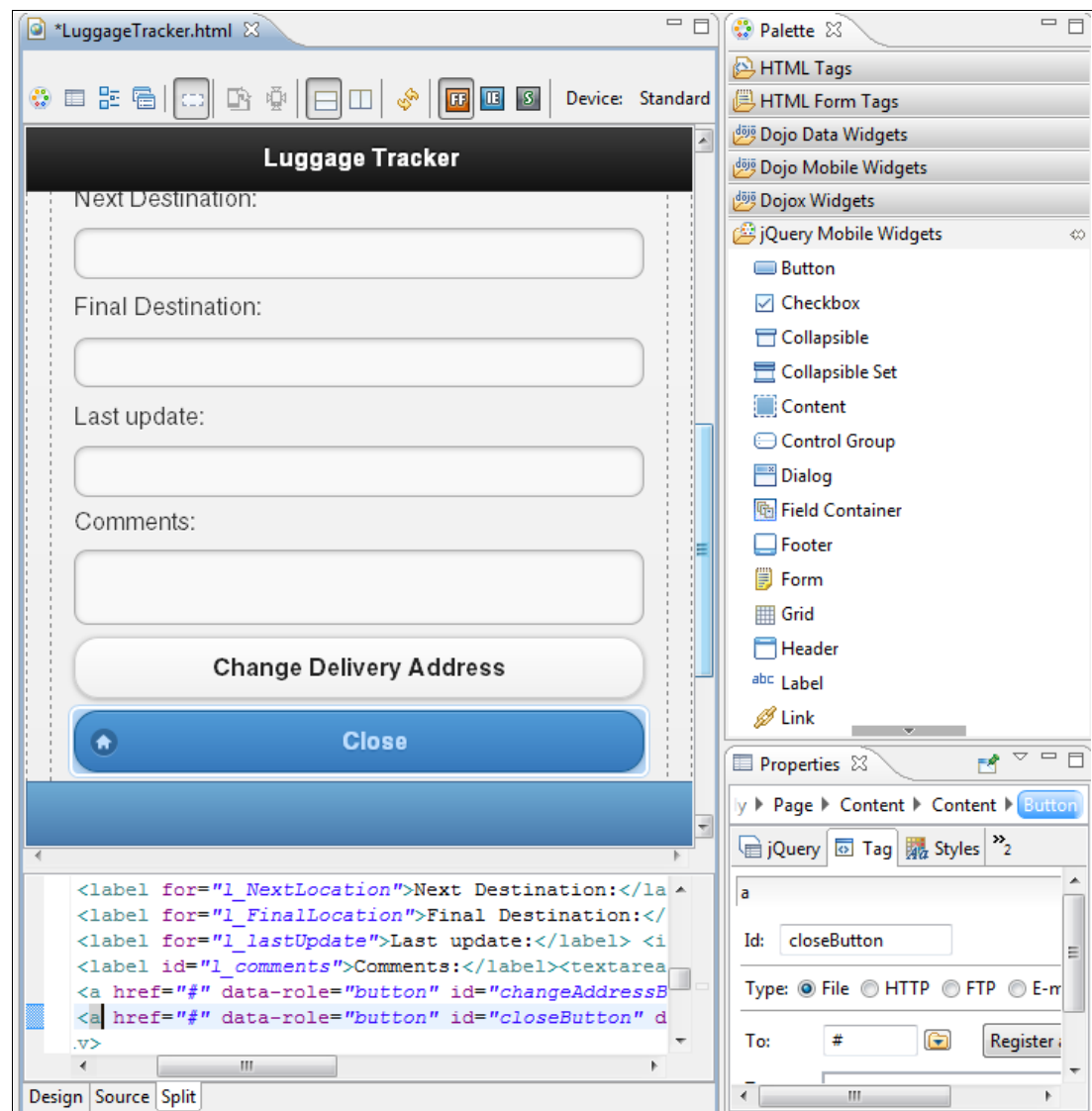


Figure 4-42 Bottom portion of finished Luggage information screen in Rich Page Editor

Login screen

Continuing in the `LuggageTracker.html` file, create the Login screen by using the same basic steps used for the previous two screens. Start by adding a Content widget, then add the needed text and button widgets and change the text, ID, and name attributes of each. The required widgets along with their label text and name and ID values are listed in Table 4-4.

Table 4-4 Label text and ID and name values for Login screen widgets

Widget type	Label text	ID and name values
Content	Not applicable	loginForm
Text Input	Username	usernameInputField
Text Input	Password	passwordInputField
Button	Login	loginButton

To hide the password in the field as the user enters it, select the Password field and then, in the Tag tab of the Properties view, set the Input type to Password. If you prefer to edit the HTML, change the type attribute for the input field (from type="text" to type="password").

When you finish, the Rich Page Editor display looks similar to Figure 4-43.

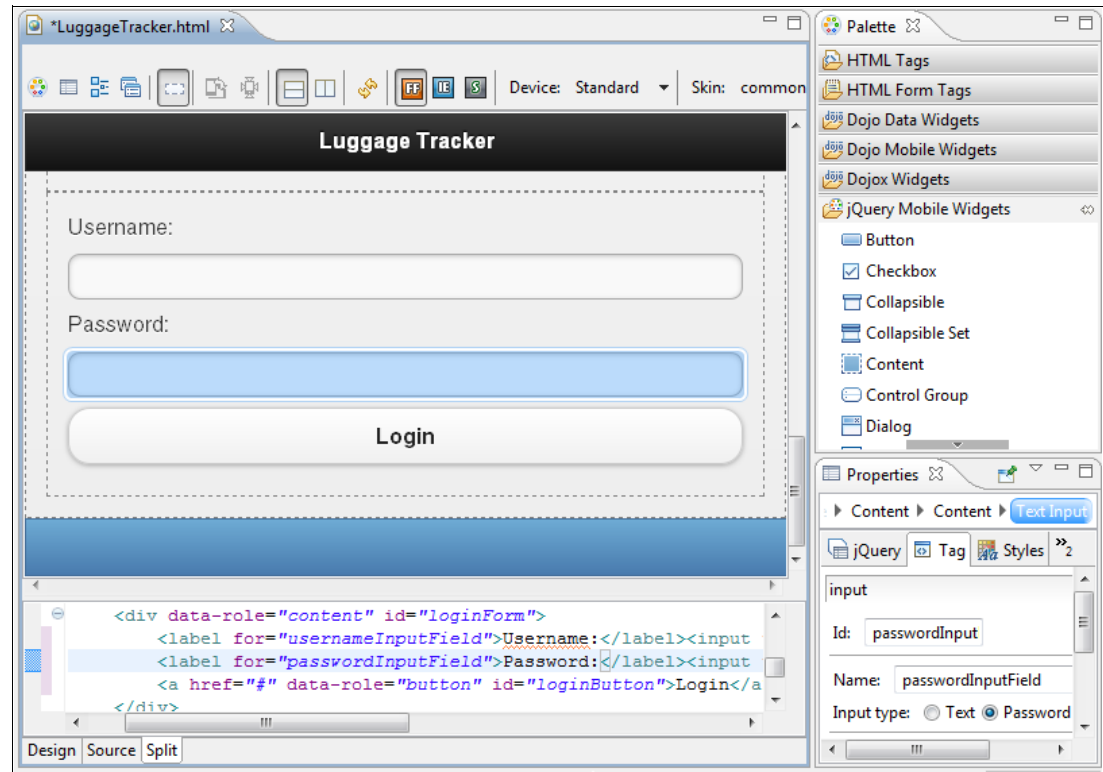


Figure 4-43 Finished Login screen in Rich Page Editor

Luggage delivery address screen

Continuing in the LuggageTracker.html file, you can also create the Luggage delivery address screen using the steps outlined previously. Start by adding a Content widget, then add the needed text input, text area, and button widgets and change the text, ID, and name attributes of each. The required widgets and their labels, names, and IDs are listed in Table 4-5.

Table 4-5 Label text and ID and name values for Luggage Delivery address screen widgets

Widget type	Label text	ID and name values
Content	N/A	setDeliveryInfo
Text Input	Luggage ID:	LuggageTrackingID
Text Input	Name:	I_DeliveryName
Text Input	Address Line 1:	I_DeliveryAdd1
Text Input	Address Line 2:	I_DeliveryAdd2
Text Input	City:	I_DeliveryCity
Text Input	State:	I_DeliveryState
Text Input	Zip Code:	I_DeliveryZip

Widget type	Label text	ID and name values
Text Input	Phone Number:	I_DeliveryPhone
Text Area	Comments	I_DeliveryComments
Button	Save and return	saveAndReturnButton
Button	Cancel	cancelButton

When you finish, your Rich Page Editor display looks similar to Figure 4-44.

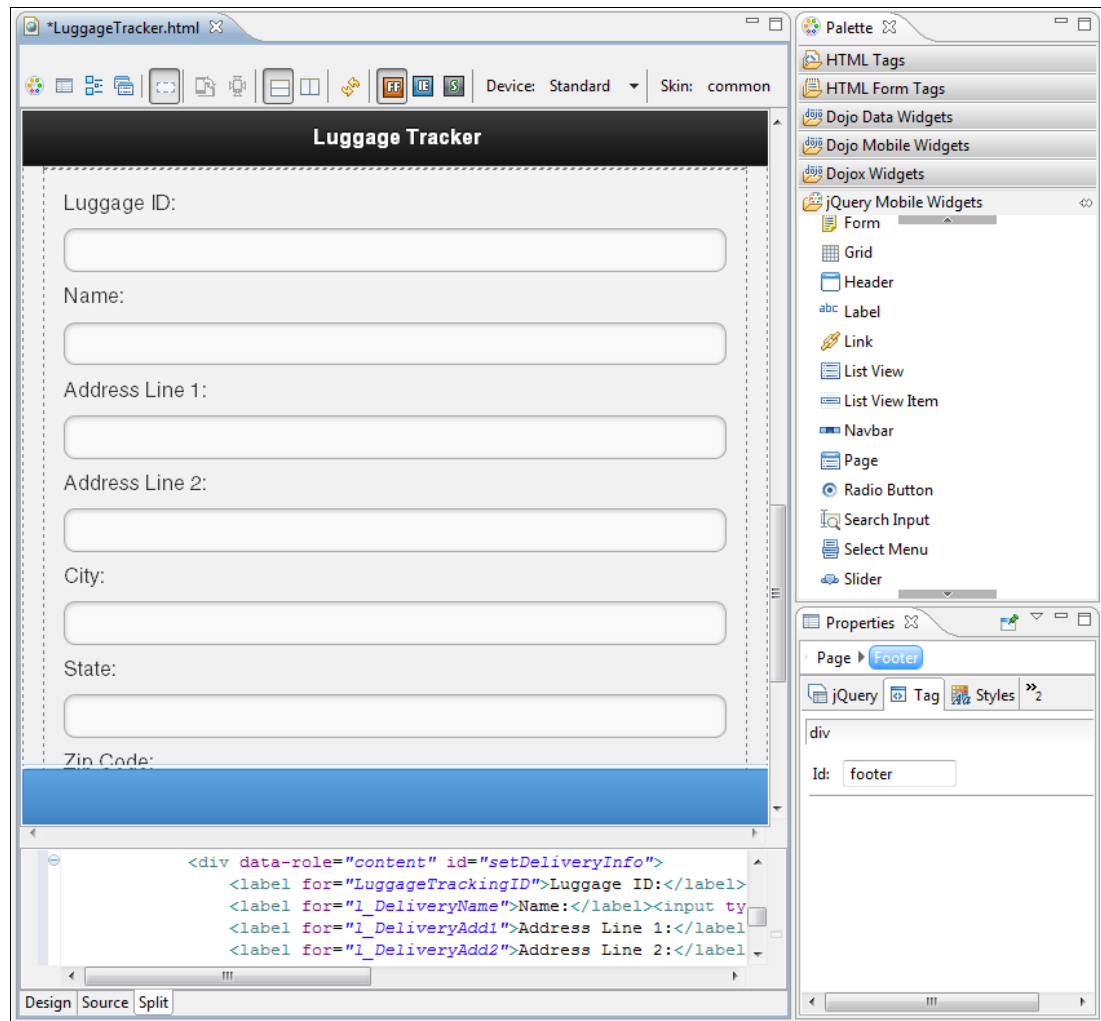


Figure 4-44 Finished Luggage delivery address screen in Rich Page Editor

Now that all four user interface screens are built, save the LuggageTracker.html file.

Hiding all but the Login screen

Only one screen will be visible on the device at a time, so all screens except the Login screen must be made hidden by default. The hiding and showing of screens is handled within the client-side JavaScript files, and screens can be hidden or shown as needed through the various user flows.

Important: On-screen buttons must eventually be linked to their respective JavaScript files. This linkage can be done in the HTML script in the Source view, or it can be done in the Design view. But if you plan to use the Design view for this task, remember that any screens you choose to hide in Rich Page Editor (which some developers do to keep their work areas more manageable) must be un-hidden before they can be modified in the Design view.

To hide the screens, use the following steps:

1. Select the Content widget for the Luggage ID input screen to highlight it.
2. In the Properties view, switch to the Styles tab.
3. Click the button on the right side of the Properties field to display the Style dialog, where you can set a variety of attributes such as font, colors, and display characteristics.
4. In the navigation pane of the Style dialog, click **Position** to display the available position properties and then select **None** from the Display drop-down list, as shown in Figure 4-45. Setting the Display property to None hides the screen.

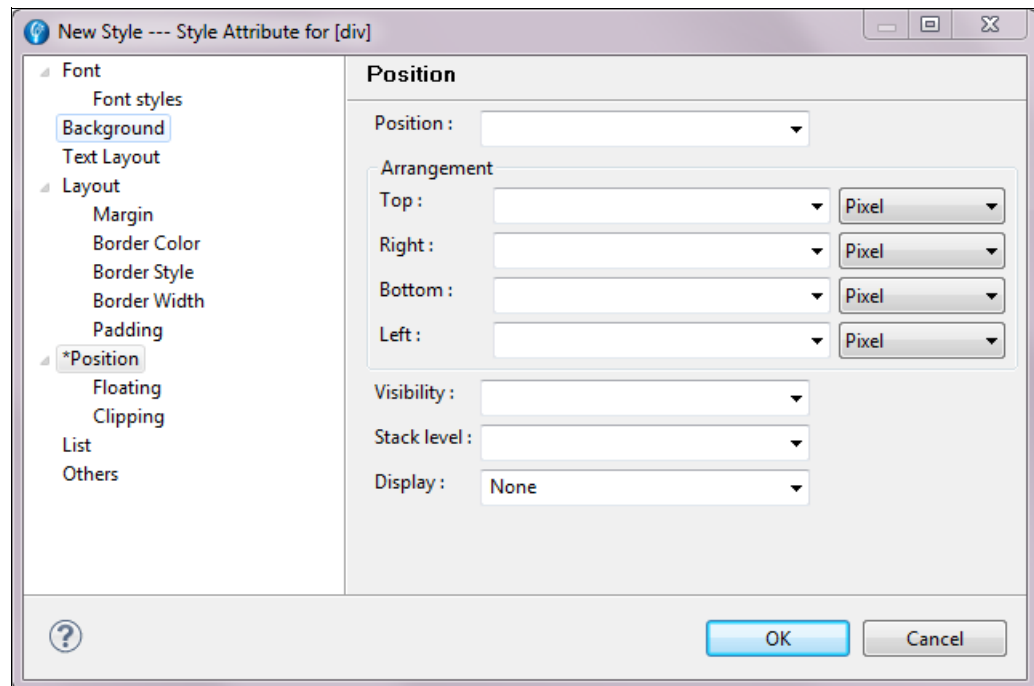


Figure 4-45 Hiding a screen using the Style dialog

5. Save these changes and close the LuggageTracker.html file.

The next step is to create the native shell for the bar code scanner component.

4.8.2 Creating the shell for the bar code scanner

A hybrid Worklight application usually consists of a container (called a *shell*) and a web application that runs in the shell (called an *inner application*). The shell provides the inner application with access to native device and mobile operating system features. This separation allows developers with different skill sets to work together to write mobile applications. An inner application developer with web expertise can create the mobile

application; a shell developer with native development skills can create the shell and provide the device functions needed for the inner application to work.

The shell can be customized to add new functions or to restrict access to native device and mobile operating system features, such as not allowing access to the device's camera. New functions within a shell can include both native device capabilities and web resources such as the company logo or JavaScript encryption libraries. These web resources can be reused by inner applications within multiple hybrid applications without having to be added to the mobile application itself.

The following sections demonstrate how to create a custom Worklight shell to add a native bar code scanner component for Android and iOS platforms. A bar code scanner component is not provided by IBM Worklight, so the LuggageTracker application will use the PhoneGap bar code scanner plug-in and that is available from GitHub:

<https://github.com/phonegap/phonegap-plugins>

The instructions in this book assume that the phonegap-plugins repository has been downloaded to your development environment. The repository folders named Android and iOS are used later in this chapter.

Other bar code scanner plug-ins can be used, but only the PhoneGap one will be shown here. If you use a different plug-in, you must determine the proper steps to integrate it into your application. To find alternate scanner plug-ins, search GitHub:

<https://github.com/>

The following high-level steps are for customizing a Worklight shell and adding native components:

1. Create a shell project.
2. Declare the common interface.
3. Add the common sources.
4. Add a new Worklight environment for each device platform to contain the platform-specific code (for example, Android).
5. Add the required sources and resources to the native project generated in step 4.
6. When tested, make the same changes in the shell template files that was done in the generated project.
7. Repeat steps 4 through 6 for each additional platform (for example, iOS).

Creating the shell project

A Worklight shell project consists of two major parts: a shell component and a shell test application. The shell component contains all of the content that is needed to build the actual shell. The test application provides the shell developer with a way to test the shell without the need to create an inner application.

To create the shell project, complete the following steps:

1. Start IBM Worklight Studio if it is not already started.
2. Click **File** → **New** → **Worklight Project** to create a new project for the shell using the New Worklight Project wizard.
3. Enter `WLAirlineShell` for the project name and choose **Shell Component** from the list of project templates.
4. Click **Next**.

5. In the Shell Component page of the wizard, enter `AirlineShellComponent` for the Component name, as shown in Figure 4-46. Because this project is for native development only, there is no need to add any of the JavaScript frameworks that were used in several projects explained previously.

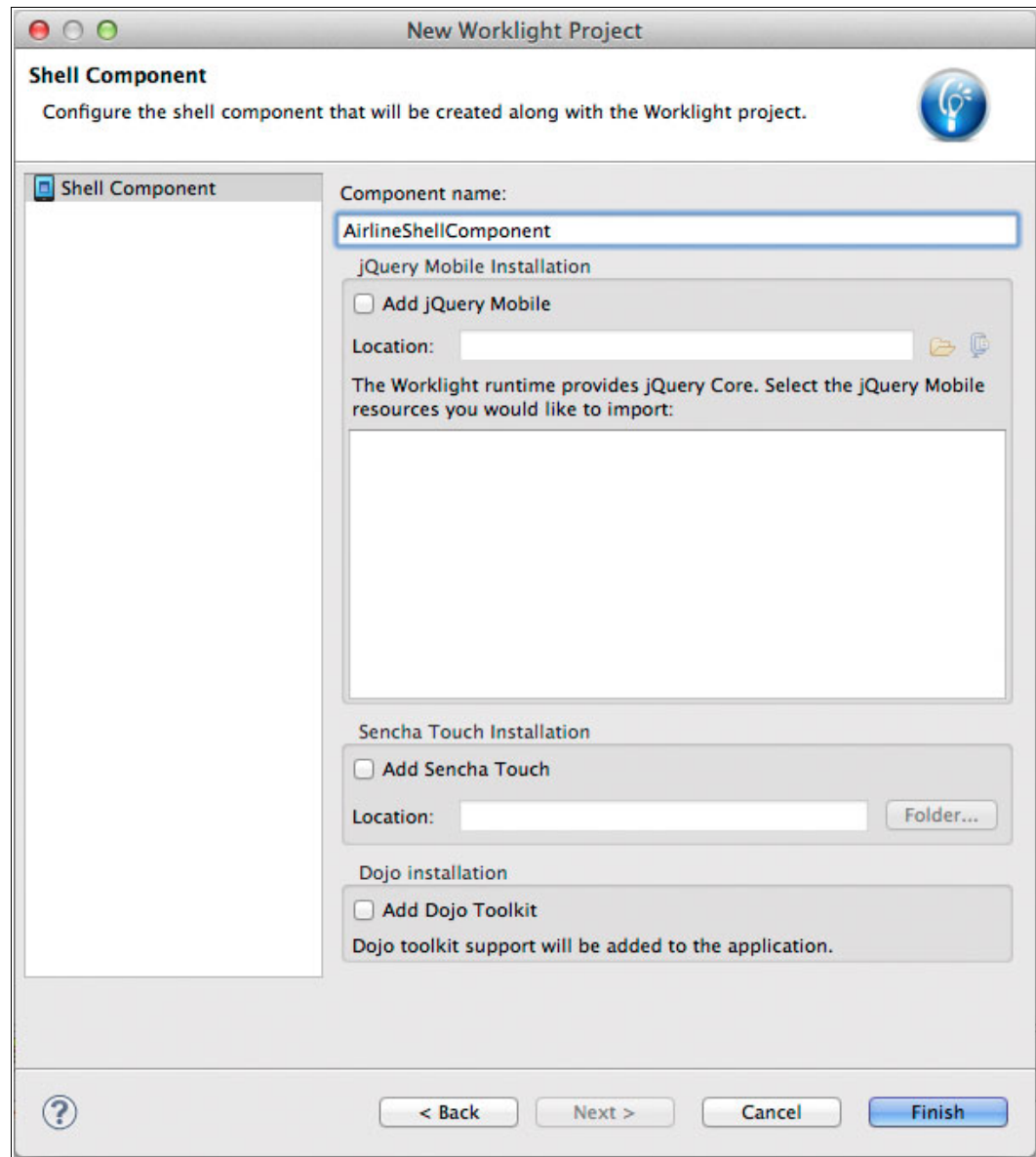


Figure 4-46 Declaring the name of the shell project component

6. Click **Finish** to create the shell project. In Project Explorer, you see a similar structure to that shown in Figure 4-47.

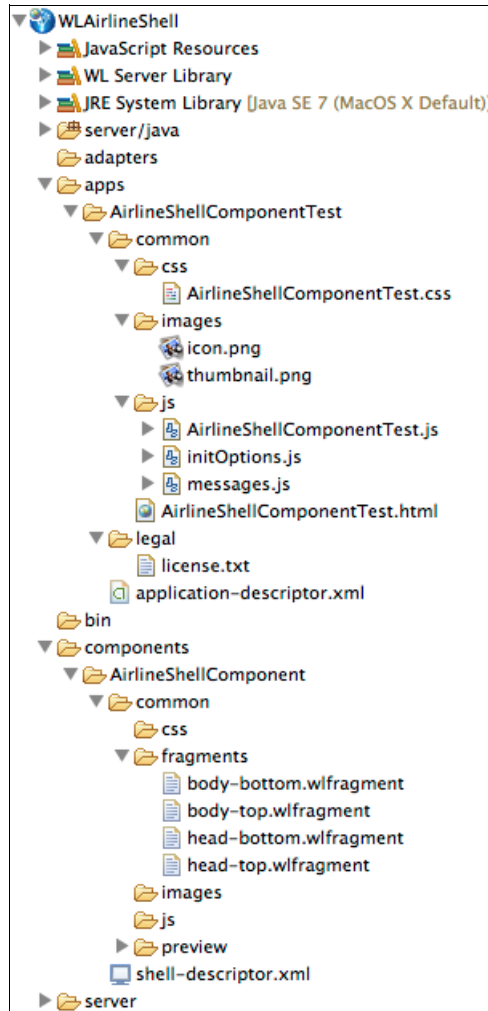


Figure 4-47 Shell project directory structure

In the shell components/AirlineShellComponent folder, the common folder includes the resources that are shared between all mobile platforms. The fragments folder contains HTML snippets that can be inserted into the main HTML file of the inner application. Typically, this is for placing commonly used HTML snippets in a single place and then selectively include them into each of the HTML pages that needs to use that HTML snippet. For example, this can be used to add a consistent header or logo throughout the application or refer to the JavaScript files that are in the shell.

In addition to the components/AirlineShellComponent folder, the following folder was also generated and contains the files for the shell-based test application:

apps/AirlineShellComponentTest/

This allows a native mobile application developer to test the shell independently, without needing to have the mobile application that will eventually use the native component inside of the shell.

To implement the bar code scanner for both Android and iOS devices, there will be some tasks that are common and must be completed regardless of how many platforms will be

supported, and tasks that are unique to a specific environment. Because LuggageTracker supports both Android and iOS devices, all of the subsections are required to complete the bar code scanner implementation.

Creating the common interface and sources

The common tasks include declaring the common interface and creating the common sources in the test application.

Declaring the common interface

When invoking native components from within a hybrid application, the current leading practice is to first define a common interface that declares the functions that can be invoked and their required parameters. For example, for scanning a bar code, you must have a scanning function that takes two callbacks as parameters, one for when the function is successful and another for when there is an error.

To create the common interface for the scanning function, use the following steps:

1. Expand the WLAirlineShell project in Project Explorer so that the components/AirlineShellComponent/common/js folder is visible.
2. Right-click the js folder and select **New** → **JavaScript source file**.
3. In the New JavaScript file dialog, enter `barcodescanner.js` as the file name and then click **Finish**.
4. The new file is displayed in the editor view, where you now must add the JavaScript for the common bar code scanner interface functions, which is shown in Example 4-16.

Example 4-16 JavaScript for bar code scanner interface

```
var BarcodeScanner = function() {};  
BarcodeScanner.prototype.scan = function(successCallback, errorCallback) {  
    if (errorCallback == null) {  
        errorCallback = function() {};  
    }  
    if (typeof errorCallback != "function") {  
        console.log("BarcodeScanner.scan failure: failure parameter not a  
function");  
        return  
    }  
    if (typeof successCallback != "function") {  
        console.log("BarcodeScanner.scan failure: success callback parameter  
must be a function");  
        return  
    }  
    this.execute(successCallback,errorCallback);  
};  
  
BarcodeScanner.prototype.execute = function(successCallback,errorCallback) {  
    errorCallback("Sorry - no Barcodescanner available");  
};  
if(!window.plugins) {  
    window.plugins = {};  
}  
if (!window.plugins.barcodeScanner) {  
    window.plugins.barcodeScanner = new BarcodeScanner();  
}
```

This new JavaScript source file is based on the file with the same name that is part of the PhoneGap bar code scanner plug-in repository used for this book. The difference between the new JavaScript file and the one in the repository is that the new JavaScript does not contain an actual call to a bar code scanner. This is the common interface and, because the bar code scanner is a native function, the execute method does nothing. When the native pieces for Android and iOS are implemented later, this execute function will be overridden by the correct platform-specific functions needed to call the bar code scanner on each platform.

5. Save and close the `barcodescanner.js` file.

The JavaScript for the scanning function in the common interface (Example 4-16 on page 125) must be referenced in the main HTML of the inner application to be able to call the scanning function. As previously mentioned, the HTML in the `fragments` folder is injected into the main HTML file of the inner application. Four fragment files are used to inject snippets into various places in the main HTML file:

- ▶ `head-top.wl` fragment: Injects the snippet at the top of the `<head>` tag
- ▶ `head-bottom.wl` fragment: Injects the snippet at the bottom of the `<head>` tag
- ▶ `body-top.wl` fragment: Injects the snippet at the top of the `<body>` tag
- ▶ `body-bottom.wl` fragment: Injects the snippet at the bottom of the `<body>` tag

In the case of the bar code scanner, the `barcodescanner.js` file must be added to the fragment file `body-top.wl` fragment to inject it into the correct place in the main HTML file.

To add the `barcodescanner.js` file to the `body-top.wl` fragment file, use these steps:

1. Expand the `WLAirlineShell` project in Project Explorer so that the files in of the `components/AirlineShellComponent/common/fragments` folder are visible.
2. Open the `body-top.wl` fragment file.
3. Immediately below the existing comments in the file, add the HTML shown in bold in Example 4-17.

Example 4-17 Fragment to load barcodescanner.js

```
<!-- This file will be injected at the top part of the <body> element. -->
<!-- Place here any HTML code, links to CSS files, and links to JavaScript
files that should be available for the application. -->
<script src="js/barcodescanner.js"></script>
```

4. Save and close the `body-top.wl` fragment file.

This will add the reference to the `barcodescanner.js` file to the main HTML file for the application during the Worklight build process, so that the functions in the `barcodescanner.js` file are available to the application.

Adding the common sources to the test application

To be able to test the bar code scanner, a call to the scan function in the common interface (Example 4-16 on page 125) must be added to the test application. To add this call, use these steps:

1. Expand the `WLAirlineShell` project in Project Explorer so that the files inside of the `apps/AirlineShellComponentTest/common/js` folder are visible.
2. Open the `AirlineShellComponentTest.js` file.

3. Add the JavaScript shown in Example 4-18 to the file in the `wlCommonInit` function, which contains the common initialization code for the test application. The additional JavaScript invokes the common interface's `scan` function.

Example 4-18 Calling the bar code scanner in the test application

```
if (window.plugins && window.plugins.barcodeScanner) {
    scanBarcode = function() {
        window.plugins.barcodeScanner.scan( function(result) {
            alert("We got a barcode\n" +
                "Result: " + result.text + "\n" +
                "Format: " + result.format + "\n" +
                "Cancelled: " + result.cancelled);
        }, function(error) {
            alert("Scanning failed: " + error);
        });
    };
} else {
    scanBarcode = function() {
        alert("No Scanner available");
    };
}
scanBarcode();
```

Note: This source code is part of the test project only and is not included in the actual shell component.

4. Save and close the `AirlineShellComponentTest.js` file.

Implementing the bar code scanner for Android

The bar code scanner is a native component and, therefore, must have a unique implementation for each platform. The following four tasks are required for completing the Android bar code scanner implementation:

- ▶ Adding a new Worklight environment
- ▶ Adding a call to the native scanning function
- ▶ Implementing and testing the bar code scanner in the test project
- ▶ Making the changes to the template files

Adding a new Worklight environment for Android

The bar code scanner component includes native source code for LuggageTracker's supported platforms, so a Worklight environment must be added for each platform.

To add a new Worklight environment for the Android platform, complete these steps:

1. Expand the `WLAirlineShell` project in Project Explorer so that the `components/AirlineShellComponent/common` folder is visible.
2. Right-click the `common` folder and select **New** → **Worklight Environment** as shown in Figure 4-48 on page 128.

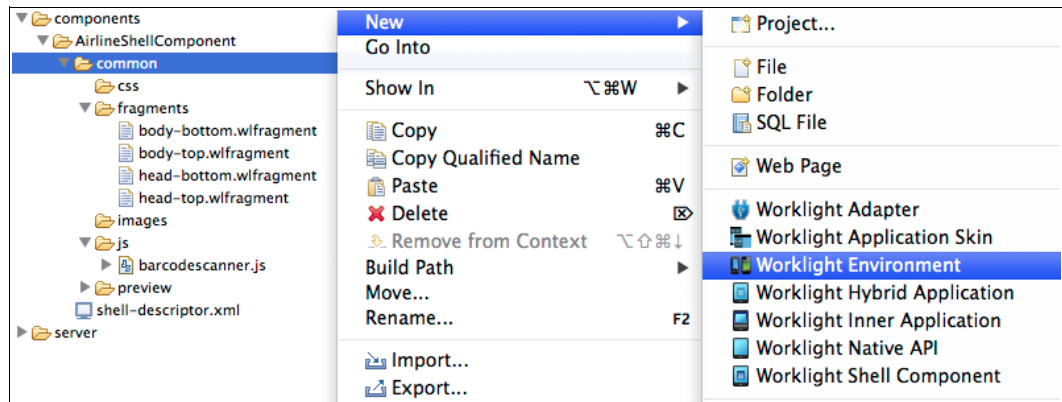


Figure 4-48 Creating a new Worklight environment for Android

3. In the New Worklight Environment dialog, the Project name and Application/Component fields are pre-filled. Because this is an environment for Android, select **Android phones and tablets**.
4. Click **Finish**.

The new Worklight Environment for Android includes two directory folders:

- ▶ components/AirlineShellComponent/android
- ▶ apps/AirlineShellComponentTest/android

The components/AirlineShellComponent/android folder has a structure similar to the one in Figure 4-49.

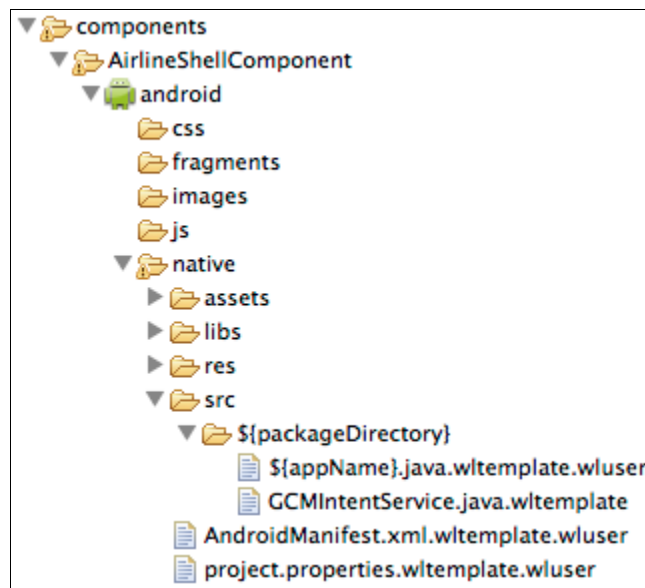


Figure 4-49 Directory structure for Android environment

Although the structure of the android folder is similar to the common folder, the android folder contains an additional folder named native. This folder and its sub-folders contain *template files* that are used to generate the native shell component and the native test project.

Examples of these template files are as follows:

- ▶ AndroidManifest.xml.wltemplate.wluser
- ▶ \${packageDirectory}/\${appName}.java.wltemplate.wluser

Notice that the second example file has an odd file name, containing the *placeholder elements* `${packageDirectory}` and `${appName}`. The file names and contents of placeholder elements are populated with the correct values during the build process. You can see an example of this in the `apps/AirlineShellComponentTest/android/native` folder, shown in Figure 4-50, which contains the generated Android project.

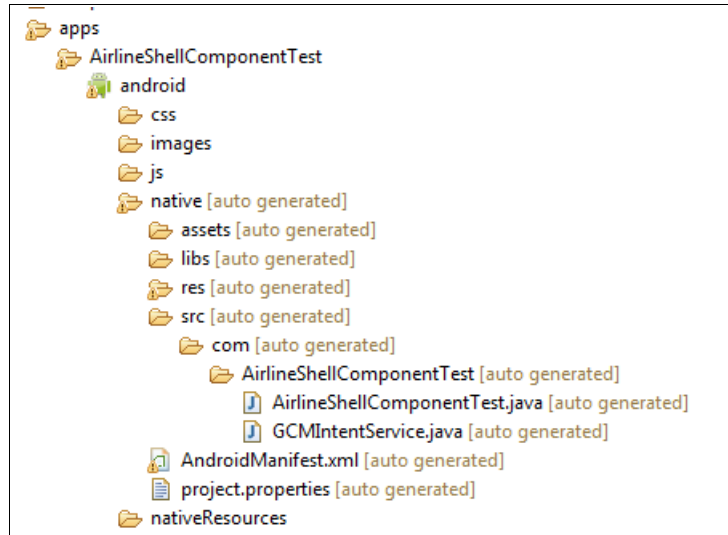


Figure 4-50 Directory structure for the generated Android environment for testing

You can see how the `${packageDirectory}` placeholder in the test component is now called `com/AirlineShellComponentTest` and the `${appName}.java.wltemplate.wluser` file name is now `AirlineShellComponentTest.java`.

In addition to the new folders created within the `WLAirlineShell` project, a new project was also generated, named `WLAirlineShellAirlineShellComponentTestAndroid`. The files in this project are linked to files in the `apps/AirlineShellComponentTest/native` folder. If the `WLAirlineShellAirlineShellComponentTestAndroid` project is ever deleted, you can re-create it by right-clicking the `apps/AirlineShellComponentTest/android` folder and then selecting **Run as** → **2. Build Environment and Deploy**.

Adding the call to the native scanning function for Android

In the `barcodescanner.js` file in the common folder, the `execute` function always returns an error, because the bar code scanner is a native plug-in and cannot run from the common folder. To overcome this limitation, each platform will have its own `barcodescanner.js` file that contains the needed calls to the native bar code scanner plug-in.

To add the native bar code scanning call for Android, complete these steps:

1. Navigate to the `components/AirlineShellComponent/android/js` folder.
2. Create a new JavaScript file by right-clicking the `js` folder and selecting **New** → **JavaScript source file**.
3. In the New JavaScript file dialog, enter `barcodescanner.js` as the file name and then click **Finish**.
4. In the Rich Page Editor, add the JavaScript code shown in Example 4-19 on page 130, which invokes the PhoneGap bar code scanner plug-in to scan a bar code (the `cordova.exec` function call in Example 4-19 on page 130). If you are using a different bar code scanner, change the line to call your particular plug-in.

Example 4-19 Function to invoke the native Android bar code scanner

```
BarcodeScanner.prototype.execute = function(successCallback,errorCallback) {  
    cordova.exec(successCallback, errorCallback, 'BarcodeScanner', 'scan', []);  
};
```

During the Worklight build process, this `barcodescanner.js` file from the `components/AirlineShellComponent/android/js` folder is merged with the `barcodescanner.js` file in the common folder.

5. Save and close the `barcodescanner.js` file.
6. Build the Android environment on the test component by right-clicking the `apps/AirlineShellComponentTest/android` environment in Project Explorer and selecting **Run as** → **Build Environment and Deploy**. You can use the Worklight Console view to monitor the progress of the build process and see any errors that occur.

Implementing and testing the bar code scanner in the test project

Up to this point, the `WLAirlineShellAirlineShellComponentTestAndroid` project that was auto-generated within the new Worklight environment for Android is empty. So, the native bar code scanner must be added to the project, and is done in two steps:

1. Implement and test the native Android bar code scanner in the test project (this section)
2. Incorporate the changes from the `WLAirlineShellAirlineShellComponentTestAndroid` project into the template files in the `AirlineShellComponent` component of the `WLAirlineShell` project (described in “Making the changes to the Android template files” on page 132).

This two-step approach is aimed at making tasks easier. These kinds of changes can be done directly in the template files in the `AirlineShellComponent` component, but the process is prone to error. An easier approach is to make the changes in the generated Android project (`WLAirlineShellAirlineShellComponentTestAndroid`), because the basic programming features of the IDE (such as auto-completion, highlighting, and the organize import function) are available when you work in the generated Android project but not when you edit the template files directly.

In the first step of the two-step approach, the actual bar code scanner plug-in files are added to the generated Android project. This involves the PhoneGap bar code scanner plug-in, which was previously downloaded to your development environment, and the `phonegap-plugins` repository.

Add the required files for the bar code scanner to the generated Android project:

1. Locate the `Android/BarcodeScanner/LibraryProject` folder inside the `phonegap-plugins` repository location where the PhoneGap bar code scanner plug-in was saved.
2. Expand the `LibraryProject` folder and copy the following folders and files from the `LibraryProject` folder to the corresponding folder in the generated Android project, making sure to not overwrite any files:
 - All files in the `src` folder
 - All files in the `res/drawable` folder
 - All files in the `res/layout` folder
 - All files in the `res/raw` folder
 - The `res/xml/preferences.xml` file

During the copy process, remember that the files in the generated Android project are linked to the `AirlineShellComponentTest` application in the `WLAirlineShell` project, so there is no directory structure for `WLAirlineShellAirlineShellComponentTestAndroid`. If possible,

drag the files from your directory explorer into the Worklight Studio Project Explorer (selecting **Copy files** on the Worklight Studio File Operation dialog that is displayed).

3. In the LibraryProject/res/values folder, rename the strings.xml file to stringsbarcode.xml. Renaming prevents you from overwriting the existing strings.xml file in the next step.
4. Copy all of the files from the LibraryProject/res/values folder to the corresponding folder in the generated Android project.
5. In Worklight Studio, refresh the Android project. Although some errors are generated, they will be resolved later.
6. Open the res/values/stringsbarcode.xml file in Rich Page Editor and delete the line containing the key labeled app_name. This step prevents duplicate key errors from occurring in the next step, where the same app_name key is encountered.
7. Save and close the stringsbarcode.xml file. This step rebuilds the project and resolves all previously generated errors except for those in the src folder.
8. The errors in the src folder are because of a missing import statement related to the Android R.java file, which is used to access a certain resource file and is generated for the package specified in the Android manifest file (AndroidManifest.xml). To fix the issue, an import statement for the com.AirlineShellComponentTest.R file must be added to each file with the error.

Add the import statement as follows:

- a. Open each file that has the import error.
- b. Right-click in the source code and select **Source** → **Organize Imports**. This step automatically adds import statements for the packages that are used in the program. If the tool cannot automatically determine which import statement to add, it presents a list of all available classes from which you can choose the correct one (in this case, select **com.AirlineShellComponentTest.R** and then click **OK**).
- c. Save and close each updated file.

After the import statement is added, the errors disappear.

9. The source files that were copied into the generated Android project in step 2 on page 130 contain activities that must be registered in the Android manifest file. To register these activities, use the following steps:
 - a. Open the AndroidManifest.xml file at the root of the generated Android project.
 - b. Switch to the tab that contains the XML source, labeled with the AndroidManifest.xml file name.
 - c. Locate the <application> tag. Within it are several <activity> tag blocks. Add the lines shown in Example 4-20 immediately beneath one of the </activity> end tags.

Example 4-20 Activity definitions to be added to the Android manifest file

```
<!-- Barcode Scanner related Activities -->
<activity android:name="com.google.zxing.client.android.CaptureActivity"
    android:screenOrientation="landscape"
    android:configChanges="orientation|keyboardHidden"
    android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
    android:windowSoftInputMode="stateAlwaysHidden"
    android:exported="false">
    <intent-filter>
        <action android:name="com.phonegap.plugins.barcodescanner.SCAN"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
```

```

</activity>
<activity
    android:name="com.google.zxing.client.android.encode.EncodeActivity"
        android:label="@string/share_name">
    <intent-filter>
        <action android:name="com.phonegap.plugins.barcodescanner.ENCODE"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>

```

d. Save your changes to the `AndroidManifest.xml` file but do not close the file.

10. The bar code scanner needs permission to use the mobile device's camera and flashlight, and these permissions also must be added to the `AndroidManifest.xml` file. Use the following steps:

- a. Near the top of the file, locate the existing `<uses-permission>` tags.
- b. Add the following lines immediately below the final `<uses-permission>` tag:

```

<!-- Barcode Scanner permissions -->
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.FLASHLIGHT" />

```

c. Save and close the `AndroidManifest.xml` file.

The project now contains the classes and resources needed to display the native Android bar code scanner plug-in, and the HTML and JavaScript files to invoke the plug-in. The only piece that is missing is the plug-in itself.

11. Add the plug-in:

- a. Following the same method used in step 2 on page 130, copy the `Android/BarcodeScanner/2.0.0/src/com` folder and its contents from the downloaded `phonegap-plugins` repository to the `src` folder in the generated Android project. The only Java file is `BarcodeScanner.java`. Do not overwrite any files.

- b. Edit the `res/xml/config.xml` file in the generated Android project to add the following line, which registers the barcode scanner plug-in, within the `<plugins>` tag:

```

<plugin name="BarcodeScanner"
    value="com.phonegap.plugins.barcodescanner.BarcodeScanner"/>

```

c. Save and close the `config.xml` file.

After these changes are completed and the bar code scanner plug-in files have been added to the generated Android project, test the application on an Android device:

1. Connect an Android device to your computer by using a USB cable.
2. In Worklight Studio, right-click the `WLAirlineShellAirlineShellComponentTestAndroid` project and select **Run as** → **1 Android Application**.
3. After the application starts, test its functionality by, for example, clicking **Scan Bar Code** to scan a bar code.

Making the changes to the Android template files

The changes made to the generated Android project now must be merged back into the template files in `AirlineShellComponent`.

The files that you previously copied into the following project, and that do not exist under `AirlineShellComponent`, can be copied directly into the folders under `AirlineShellComponent`:

`WLAirlineShellAirlineShellComponentTestAndroid`

However, for files that already exist under `AirlineShellComponent`, the modified files from the project must be merged into the `AirlineShellComponent` folder manually, as follows:

1. Using the local directory explorer, locate the `Android/BarcodeScanner/LibraryProject` folder inside the downloaded `phonegap-plugins` repository.
2. Copy the files that do not already exist under `AirlineShellComponent` by expanding the `LibraryProject` folder and then copying the following folders and files from that folder to the corresponding locations in the `components/AirlineShellComponent/android/native` folder of the `WLAirlineShell` project. Do not overwrite any files:
 - All files in the `src` folder
 - All files in the `res/drawable` folder
 - All files in the `res/layout` folder
 - All files in the `res/raw` folder
 - All files in the `res/values` folder
 - The `res/xml/preferences.xml` file
 - The `Android/BarcodeScanner/2.0.0/src/com` folder and its contents from the downloaded `phonegap-plugins` repository
3. Do the file and folder merges as follows:
 - a. In Worklight Studio, open both the `AndroidManifest.xml` file in the `WLAirlineShellAirlineShellComponentTestAndroid` project and the `AndroidManifest.xml.wltemplate.wluser` file in the `WLSHELL` project.
 - b. Copy the two activities for the bar code scanner that you added in Step 9 on page 131, and the two bar code permissions that you added in Step 10 on page 132, from the `AndroidManifest.xml` file into the `AndroidManifest.xml.wltemplate.wluser` file.
 - c. Save the `AndroidManifest.xml.wltemplate.wluser` file and then close both files.
 - d. Open the `config.xml` file in the `WLAirlineShellAirlineShellComponentTestAndroid` project and the `config.xml.wluser` file in the `WLSHELL` project.
 - e. Copy the bar code scanner `<plugin>` tag line that was added to the `config.xml` file in step 11 on page 132 to the same location in the `config.xml.wluser` file.
 - f. Save the `config.xml.wluser` file and close both files.

Worklight Studio is now able to rebuild the generated Android project. However, one problem remains regarding the `R.java` file. In the generated Android project, you changed the import statements to use the `R.java` file in the `AirlineShellComponentTest` application. But that was only the testing application. Because the final shell component will be regenerated by the Worklight build process, any import statements in the Java files of the `AirlineShellComponent` folders must use Worklight placeholders. These placeholders appear in file names and file contents and are eventually replaced by the correct package name during the Worklight build process. In addition to changing the import statements within the Java files, the name of each file that contains a placeholder must also be changed to have the following suffix, which is necessary for correct processing during the Worklight build process

`.wltemplate.wluser`

For platforms other than Mac, these changes must be made manually. If you are using Mac OS X, the script shown in Example 4-21 on page 134 can change all occurrences of `import com.AirlineShellComponentTest.R` to `import ${packageName}.R`, and also add the `.wltemplate.wluser` suffix to the file name.

Example 4-21 Script to replace specific import statements on Mac OS X

```
#!/bin/sh

# The aim of this shell script is to replace all occurrences of 'import
[javaPackageName].R' with 'import ${packageName}.R'

if [ $# != 1 ] ; then
    echo "Supply only one package name we need to look for in all java files!"
    echo "Usage: $0 PackageNameToSearchFor"
    exit 1
fi

echo looking for $1

for i in `find . -iname "*.java" -exec grep -l "import $1.R" {} \`;`
do
    sed "s/import $1.R/import \${packageName}.R/g" "$i" > "$i.wltemplate.wluser"
    echo $i change and renamed to $i.wltemplate.wluser
    rm $i
done
```

To use the script in Example 4-21, create a new file named `replacePackageAndRename.sh` in the `components/AirlineShellComponent/android/native/src` folder. Then, copy the script lines into the file, save the file, and invoke the script from a Mac OS X terminal window using the following command:

```
./replacePackageAndRename.sh com.AirlineShellComponentTest
```

When the script completes, refresh the project to see the changes. The targeted files now have the required `.wltemplate.wluser` suffix and the import statements are changed appropriately.

After the changes are confirmed, rebuild the Android project by right-clicking `apps/AirlineShellComponentTest/android` and then selecting **Run as → Build Environment and Deploy**.

Implementing the bar code scanner for iOS

As with the Android bar code scanner, an implementation for iOS also must be created. The following four tasks are required for completing the iOS bar code scanner implementation:

- ▶ Adding a new Worklight environment
- ▶ Adding a call to the native scanning function
- ▶ Implementing and testing the bar code scanner in the test project
- ▶ Making the changes to the template files

Add a new Worklight environment for iOS

First, a Worklight environment for iOS must be added. However, unlike when you added an environment for the Android version of the application, these steps do not result in the creation of a new Worklight project. The reason is because native applications for iOS are edited and tested using Xcode, an IDE with software development tools specifically for OS X and iOS environments.

Add the needed environment for iOS:

1. In Worklight Studio, expand the WLAirlineShell project in Project Explorer so that the components/AirlineShellComponent folder is visible.
2. Right-click the components/AirlineShellComponent folder and select **New** → **Worklight Environment**.
3. Make sure the correct project and component are selected and then choose **iPhone** and click **Finish**.

Two new folders named iphone are now created:

- ▶ One under components/AirlineShellComponent
- ▶ One under apps/AirlineShellComponentTest

The structure of these folders is similar those in the Android environment, but the contents differ because they contain the template files to create an Xcode project for iOS when the application is opened in the Xcode IDE.

To edit or test the application, open the Xcode project by right-clicking the apps/AirlineShellComponentTest/iphone/native folder and selecting **Run as** → **5. Xcode Project**, which launches the Xcode IDE. The generated native iOS test application is available in the apps/AirlineShellComponentTest/iphone/native folder.

Adding the call to the native scanning function for iOS

Similar to the Android application, the bar code scanner plug-in cannot be invoked on an iOS device until there is a working execute function that calls a native bar code scanner. Remember that the common barcodescanner.js file is the common interface and each environment will have a barcodescanner.js file that contains the native implementation.

Create the barcodescanner.js for iOS and add the native bar code scanning call by completing the following steps:

1. Navigate to the components/AirlineShellComponent/iphone/js folder.
2. Create a new JavaScript file by right-clicking the js folder and then selecting **New** → **JavaScript source file**.
3. In the New JavaScript file dialog, enter barcodescanner.js as the file name and then click **Finish**.
4. Add the JavaScript code shown in Example 4-22, which invokes the PhoneGap bar code scanner plug-in to scan a bar code (the cordova.exec function call in Example 4-22). If you are using a different bar code scanner plug-in, you must change the line to call your particular plug-in.

Example 4-22 JavaScript code to invoke the native iOS bar code scanner

```
BarcodeScanner.prototype.execute = function(successCallback,errorCallback) {  
    cordova.exec(successCallback, errorCallback,  
        'org.apache.cordova.barcodeScanner', 'scan', []);  
};
```

5. Save and close the barcodescanner.js file.

When you finish, rebuild the project by right-clicking the apps/AirlineShellComponentTest/iphone folder and selecting **Run as** → **2 Build Environment and Deploy**.

Implementing and testing the bar code scanner in the test project

As with the Android project, the iOS component is initially empty. To add bar code scanning functions, certain required Objective-C classes, frameworks, and libraries must be added. This is easier to do in the Xcode project (using the Xcode IDE) than in the templates directly.

To add the required Objective-C classes, frameworks, and libraries, use the following steps:

1. Open the Xcode IDE by right-clicking apps/AirlineShellComponentTest/iphone and then selecting **Run as** → **5. Xcode Project**.
2. In the Xcode project, right-click the **Classes** folder and then select **Add Files to WLAirlineShellAirlineShellComponentTestIphone**.
3. In the dialog that is opens, select the CDVBarcodeScanner.mm, zxing-all-in-one.cpp and zxing-all-in-one.h files from the iOS/BarcodeScanner folder of your phonegap-plugins repository. Then choose **Copy items into destination group's folder (if needed)** (so that the files are copied into your project and not merely linked to it), and then click **Add**.
4. The following three frameworks and one library, which are required by the PhoneGap bar code scanner plug-in, must be added to the project in Xcode:
 - AVFoundation.framework
 - AssetsLibrary.framework
 - CoreVideo.framework
 - libiconv.dylib

To add the frameworks and library, use the following steps:

- a. Select the WLAirlineShellAirlineShellComponentTestIphone project and then choose the WLAirlineShellAirlineShellComponentTestIphone target, as shown in Figure 4-51.

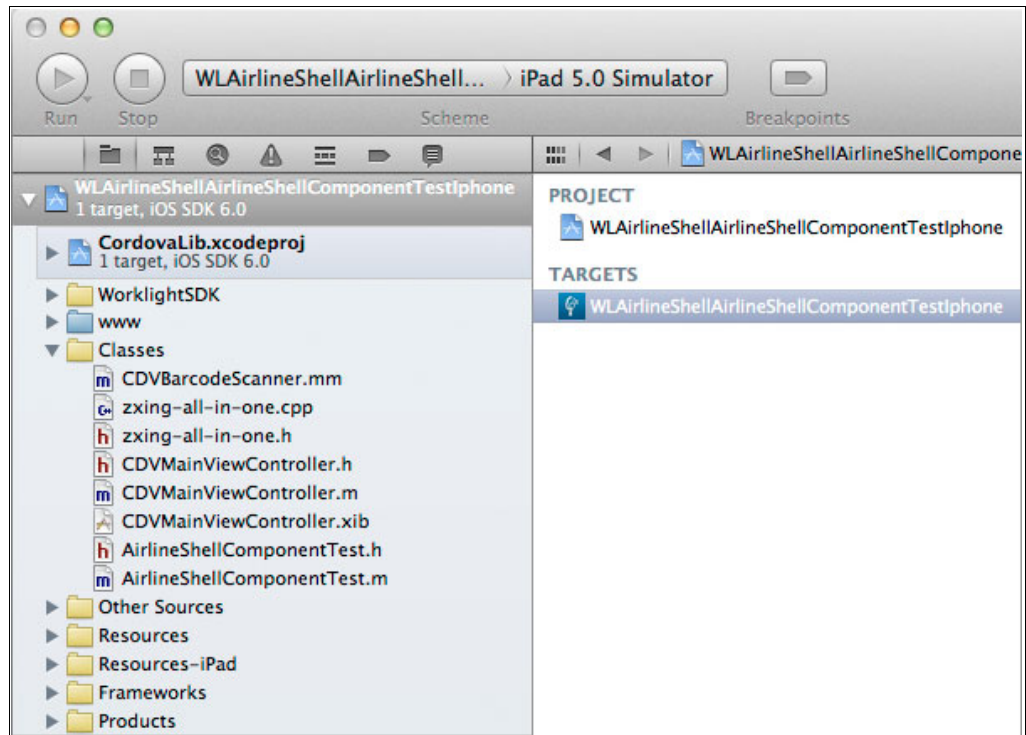


Figure 4-51 Selecting the project and target in the Xcode IDE

- b. Click **Build Phases** (located in the bar to the right of the PROJECT and TARGETS section).

- c. Expand the **Link Binary With Libraries** section.
- d. Click the plus sign (+) in the lower-left corner of the section to open a dialog that lists available frameworks and libraries.
- e. Select each of the required frameworks, as shown in Figure 4-52, and then click **Add**.
- f. Select the required library and then click **Add**.

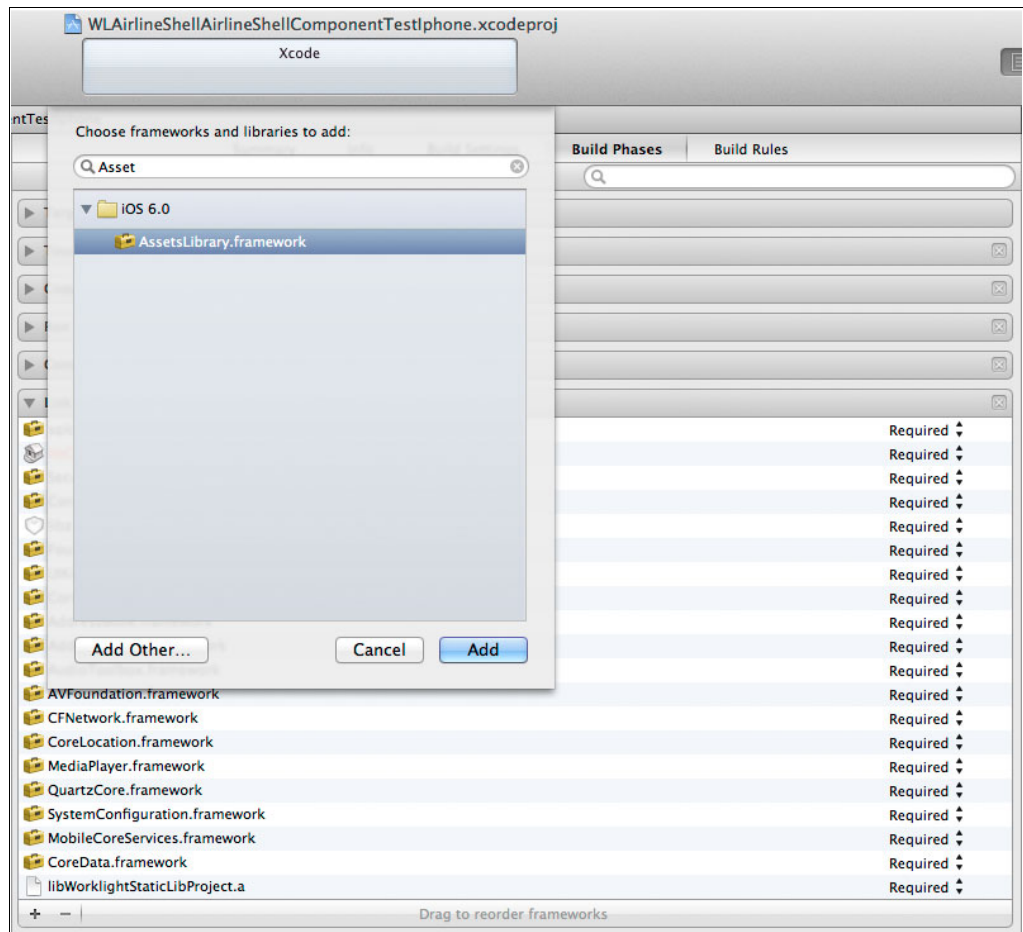


Figure 4-52 Adding frameworks and libraries to the iOS project

The project now contains the classes, frameworks, and libraries needed to display the native iOS bar code scanner when it is called from the plug-in, and the HTML and JavaScript files to invoke the plug-in. Now you must register the plug-in:

1. Expand the Resources folder in your Xcode project and open the `Cordova.plist` file.
2. In the `.plist` file, expand Plugins and then click the plus sign (+) that is displayed when you hover the mouse over the Plugins line.
3. Enter `org.apache.cordova.barcodeScanner` for the key and `CDVBarcodeScanner` for the value, as shown in Figure 4-53.



Figure 4-53 Cordova.plist file with the registered bar code scanner plug-in

4. Save and close the `Cordova.plist` file.

The bar code scanner is now enabled when the application is run on an iOS device with a camera, similar to the way the Android version works. You can now test the bar code scanning function on an iOS device.

Making the changes to the iOS template files

Because the changes to enable the bar code scanner were done in the generated native iOS project, the changes must be applied back to the template files. Follow these steps:

1. Copy the CDVBarcodeScanner.mm, zxing-all-in-one.cpp and zxing-all-in-one.h files from the iOS/BarcodeScanner folder of your phonegap-plugins repository to the components/AirlineShellComponent/iphone/native/Classes folder.
2. Open the components/AirlineShellComponent/iphone/native/Cordova.plist.wluser file.
3. Copy the two lines from Example 4-23 into the following file by placing them in the <dict> tag below the line that says <key>Plugins</key>:

components/AirlineShellComponent/iphone/native/Cordova.plist.wluser

Example 4-23 Registering the bar code scanner plug-in in the Cordova.plist file

```
<key>org.apache.cordova.barcodeScanner</key>
<string>CDVBarcodeScanner</string>
```

4. Save and close the Cordova.plist.wluser file.

Because Xcode stores all information about added files, libraries, and frameworks in a project file, that project file also must be updated. If this task is not done, the added source files, libraries, and frameworks will not be included in the Xcode project when it is regenerated.

To merge the changes from the Xcode project's project file with AirlineShellComponent, use the following steps:

1. Close Xcode if it is still open.
2. Make a backup copy of the following file:
apps/AirlineShellComponentTest/iphone/native/WLAirlineShellAirlineShellComponentTestIphone.xcodeproj/project.pbxproj
3. Make a backup copy of the following file:
components/AirlineShellComponent/iphone/native/\${xcodeProjectName}.xcodeproj.wluser/project.pbxproj.wltemplate.wluser
4. Merge the changes contained in these two files into AirlineShellComponent:
 - a. Open Xcode and then open its FileMerge application by selecting **Xcode** → **Open Developer Tool** → **FileMerge**.
 - b. Click **Left**, select the apps/AirlineShellComponentTest/iphone/native/WLAirlineShellAirlineShellComponentTestIphone.xcodeproj file, and then click **Open**. In the text box that opens, append /project.pbxproj to the end of the displayed file name.
 - c. Click **Right** and select the components/AirlineShellComponent/iphone/native/\${xcodeProjectName}.xcodeproj.wluser/project.pbxproj.wltemplate.wluser file.
 - d. Click **Compare** and then look carefully at the differences between the two files.
 - e. For each item that is different, you must specify whether the merged file will use what is shown in the file on the left side of the display or what is shown in the file on the right side of the display. You do this by selecting a particular change (indicated by a

numerical number in the center of the FileMerge application, as shown in Figure 4-54) and then selecting **Choose left** or **Choose right** from the Actions menu. Ensure that the Worklight placeholders (for example, \${projectName}) are included in the resulting merged file, along with the files you added in step 1 on page 138 and the required three frameworks and one library.

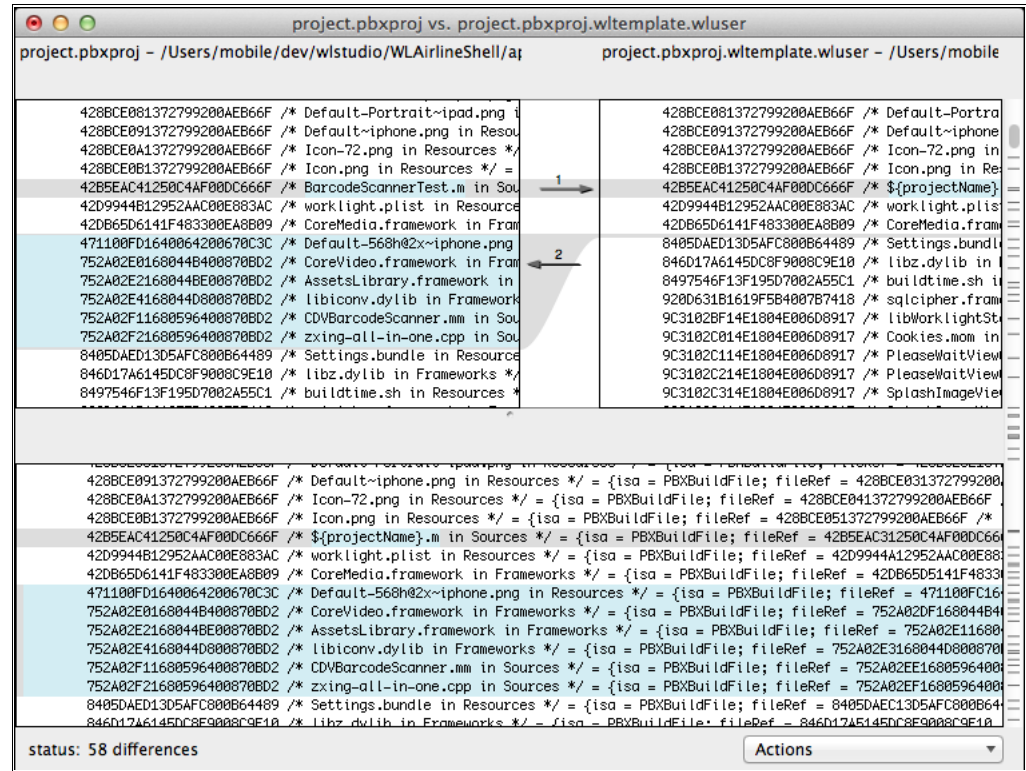


Figure 4-54 Merging two Xcode project files in the FileMerge application

- f. Select **File** → **Save Merge As** and save the merged file as follows:
`components/AirlineShellComponent/iphone/native/${xcodeProjectName}.xcodeproj.wluser/project.pbxproj.wltemplate.wluser.`
- g. Close the FileMerge application.
5. In Worklight Studio, refresh the components folder under the WLAirlineShell project.
6. Rebuild the iOS project by right-clicking apps/AirlineShellComponentTest/iphone and then choosing **Run as** → **5 Xcode Project**.

Building the Worklight shell

After the Worklight shell is created and the bar code scanner component is added for both the Android and iOS platforms, build the Worklight Shell project to generate a shell file. This shell file will then be provided to the mobile web application developers who will use the APIs defined within the shell component as part of their hybrid application.

To build the shell, use the following steps:

1. Expand the WLAirline project and expand the components folder.
2. Right-click the AirlineShellComponent folder and then select **Run as** → **1 Build Shell component**. This starts the build process of the shell component.

When the build completes, the new `AirlineShellComponent-1.0.wlshell` file is in the `bin` folder of the `WLAirlineShell` project and is ready to be used in building the application (see 4.8.6, “Building the mobile application” on page 160).

4.8.3 Creating the client-side authentication component

The client-side authentication component, as outlined in 4.4, “Planning for security” on page 75, consists of a login screen to prompt the user for credentials and a challenge handler to perform the client-side authentication logic.

When the application starts, the Login screen is displayed to gather the CSR’s credentials. The credentials are then authenticated and, if this is successful, the Luggage ID input screen is displayed. This process is shown in Figure 4-10 on page 76.

As explained when the user interface screens were created, each screen is defined within a Content widget and each widget has a unique ID. Within the `LuggageTracker.html` file, the following two content widgets are defined in HTML `<div>` tags:

- ▶ The `<div id=“loginForm”>` element for the Login screen
- ▶ The `<div id=“luggageIdInput”>` element for the Luggage ID input screen

The client-side authentication procedure (see 4.4.2, “Planning client-side authentication” on page 76) is implemented in Worklight Studio, in a new file named `auth.js` that is located in the `apps/common/js` folder of the `LuggageTracker` application. Use the following steps to create this JavaScript file:

1. Start Worklight Studio if it is not already started.
2. Expand the `CompanyAirlines` project until the `apps/LuggageTracker/common/js` folder is visible.
3. Create a new JavaScript file by right-clicking the `js` folder and then selecting **New** → **JavaScript source file**.
4. In the New JavaScript file dialog, enter `auth.js` as the file name and then click **Finish**.
The next several steps involve adding JavaScript source code to this file.
5. Create the `loginChallengeHandler` for the `LuggageTrackerRealm` (described in 4.7.2, “Creating the authentication adapter” on page 91) by adding the following line to the new `auth.js` file:

```
var loginChallengeHandler =  
WL.Client.createChallengeHandler("LuggageTrackerRealm");
```

6. Create the Authenticator by appending the lines in Example 4-24 on page 141 to end of the new `auth.js` file. Make sure that the ID and name for the Username and Password fields of the Login screen match the names shown in bold in the example.

Several important actions are handled within the Authenticator:

- Instead of connecting the button to a function by using Rich Page Editor, the JavaScript binds the click action of the Login button to an internal function that is defined within the Authenticator function.
- Using the same method, the click action of the Logout button is bound to the `onLogout` function.
- When the Login button is clicked, the internal function within the Authenticator gathers the values entered into the user interface fields and passes these values as parameters to `LuggageTrackerAdapter`’s `submitAuthentication` procedure.

Example 4-24 JavaScript for the Authenticator function

```
var Authenticator = function () {
    $("#loginButton").bind('click', function () {
        var username = $("#usernameInputField").val();
        var password = $("#passwordInputField").val();

        var invocationData = {
            adapter : "LuggageTrackerAdapter",
            procedure : "submitAuthentication",
            parameters : [ username, password ]
        };
        WL.Logger.debug("Password : " + password);
        WL.Logger.debug("Username: " + username);
        loginChallengeHandler.submitAdapterAuthentication(invocationData, {});
    });
    $("#logoutButton").bind('click', onLogout);
    if(WL.Client.isUserAuthenticated("LuggageTrackerRealm")) {
        onLogout();
    }
}();
```

7. Append the lines shown in Example 4-25 to the end of the new `auth.js` file. These lines handle the response that is returned from the adapter's `submitAuthentication` procedure.

Based on the response, the `loginChallengeHandler` component either invokes the `authFail` function (if unsuccessful) or the `loadApp` function (if successful).

Example 4-25 JavaScript to handle the authentication response

```
loginChallengeHandler.isCustomResponse = function(response){
    if (!response || !response.responseJSON || response.responseText === null)
        return false;
    if (typeof(response.responseJSON.authRequired) != 'undefined'){
        return true;
    } else {
        return false;
    }
};

loginChallengeHandler.handleChallenge = function(response){
    var authRequired = response.responseJSON.authRequired;
    if (authRequired){
        authFail();
    } else {
        loadApp(response);
        loginChallengeHandler.submitSuccess();
    }
};
```

8. Next, add the `loadApp` and `authFail` functions, shown in Example 4-26 on page 142, to the bottom of the new `auth.js` file.

The `authFail` function adds a debug line to the Worklight log and redisplay the Login screen. The `loadApp` function displays the Luggage ID input screen.

Example 4-26 JavaScript source for the loadApp and authFail functions

```
function loadApp(response) {
    var isSuccessful = response.responseJSON.isSuccessful;
    WL.Logger.debug("login successful:" + isSuccessful);
    var authRequired = response.responseJSON.authRequired;
    WL.Logger.debug("authRequired value: " + authRequired);
    if(authRequired){
        $("#luggageIdInput").hide();
        $("#loginForm").show();
    } else{
        $("#luggageIdInput").show();
        $("#loginForm").hide();
    }
}
function authFail() {
    WL.Logger.debug("Log In Failed");
    WL.Client.reloadApp;
}
```

9. Finally, add the Logout function shown in Example 4-27 to the end of the new auth.js file.

Example 4-27 JavaScript source for the Logout function

```
function Logout(){
    var invocationData = {
        adapter : "LuggageTrackerAdapter",
        procedure : "onLogout",
        parameters : []
    };
    loginChallengeHandler.submitAdapterAuthentication(invocationData, {});
    WL.Logger.debug("Logged out");
    $("#luggageIdInput").hide();
    $("#loginForm").show();
}
```

10. Save and close the auth.js file.

The complete contents of the updated auth.js file are shown in “Source for auth.js” on page 347.

4.8.4 Integrating the solution

At this point, four distinct parts of the mobile application must be integrated:

- ▶ The user interface, created in 4.8.1, “Creating the user interface” on page 105
- ▶ The bar code scanner component, detailed in 4.8.2, “Creating the shell for the bar code scanner” on page 121
- ▶ The client-side security component, created in 4.8.3, “Creating the client-side authentication component” on page 140
- ▶ The back-end business services adapter, created in 4.7.3, “Creating the business services adapter” on page 99

Integrating the bar code scanner component

At the end of 4.8.2, “Creating the shell for the bar code scanner” on page 121, a shell component file, `AirlineShellComponent-1.0.wlshell`, was created. Now the mobile web application developer integrates this shell component into the mobile application by using the following steps:

1. Open Worklight Studio if it is not already open.
2. Expand the `CompanyAirlines` project until the `apps/LuggageTracker` folder is visible
3. Open the `application-descriptor.xml` file.
4. On the Source tab, locate the line with the `<thumbnailImage>` tag and add the following lines (which define the location of the `wlshell` file) immediately below it:

```
<shell>
  <archive>../../wlshell/AirlineShellComponent-1.0.wlshell</archive>
</shell>
```

You can also change the description and author information in this file, if you want to.

The updated contents of the `application-descriptor.xml` file are shown in Example 4-28, with the newly added lines highlighted in bold. For clarity, some comments in the file have been removed from the example.

Example 4-28 LuggageTracker application description XML

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Attribute "id" must be identical to application folder name -->
<application id="LuggageTracker" platformVersion="5.0.5"
  xmlns="http://www.worklight.com/application-descriptor"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <displayName>LuggageTracker</displayName>
  <description>LuggageTracker</description>
  <author>
    <name>application's author</name>
    <email>application author's e-mail</email>
    <copyright>Copyright My Company</copyright>
    <homepage>http://mycompany.com</homepage>
  </author>
  <height>460</height>
  <width>320</width>
  <mainFile>LuggageTracker.html</mainFile>
  <thumbnailImage>common/images/thumbnail.png</thumbnailImage>
  <shell>
    <archive>../../wlshell/AirlineShellComponent-1.0.wlshell</archive>
  </shell>

  <worklightServerRootURL>http://${local.IPAddress}:8080</worklightServerRootURL>

</application>
```

5. Save and close the `application-descriptor.xml` file.

Next, create the bar code scanner JavaScript call that will invoke the native bar code scanner component, which is used to retrieve the luggage identifier number. Use the following steps:

1. Open the `LuggageTracker.js` file located within the `apps/LuggageTracker/common/js` folder.
2. Add the `scanBarcode` function shown in Example 4-29 to the file, placing it beneath the `wlCommonInit` function.

The `scanBarcode` function verifies the existence and availability of the bar code scanner plug-in and, if it exists, calls the plug-in and returns the scanned luggage identifier number. If the bar code scanner plug-in does not exist, a message is displayed to the user stating that the bar code scanner is not available.

Example 4-29 JavaScript for the scanBarcode function

```
function scanBarcode() {

    if (window.plugins && window.plugins.barcodeScanner) {
        scanBarcode = function() {
            window.plugins.barcodeScanner.scan(function(result) {
                if (!result.cancelled) {
                    $("#LuggageTrackingID").val(result.text);
                    document.getElementById("notifications").innerHTML = "";
                    WL.Client.logActivity("Bar Code Scanner scanned the bar code
successfully ");
                }
            }, function(error) {
                document.getElementById("notifications").innerHTML = error;
                WL.Client.logActivity("Bar Code Scanner Error "+error);
            });
        };
    } else {
        scanBarcode = function() {
            document.getElementById("notifications").innerHTML = "No Scanner
available";
            WL.Client.logActivity("No Scanner available");
        };
    }

    scanBarcode();
}
```

3. Save and close the `LuggageTracker.js` file.

Finally, connect the Scan Bar Code button to the `scanBarcode` function as follows:

1. Open the `LuggageTracker.html` file within the `apps/LuggageTracker/common` folder.
2. Open the Properties view for the Scan Bar Code button, either by selecting the button (if the screen is not hidden) or selecting the button HTML tag on the Source tab.
3. In the Properties view, go to the All tab and locate the button's onclick property.

4. Click in the **Value** column of the onclick property and enter `scanBarCode()`, as shown in Figure 4-55.

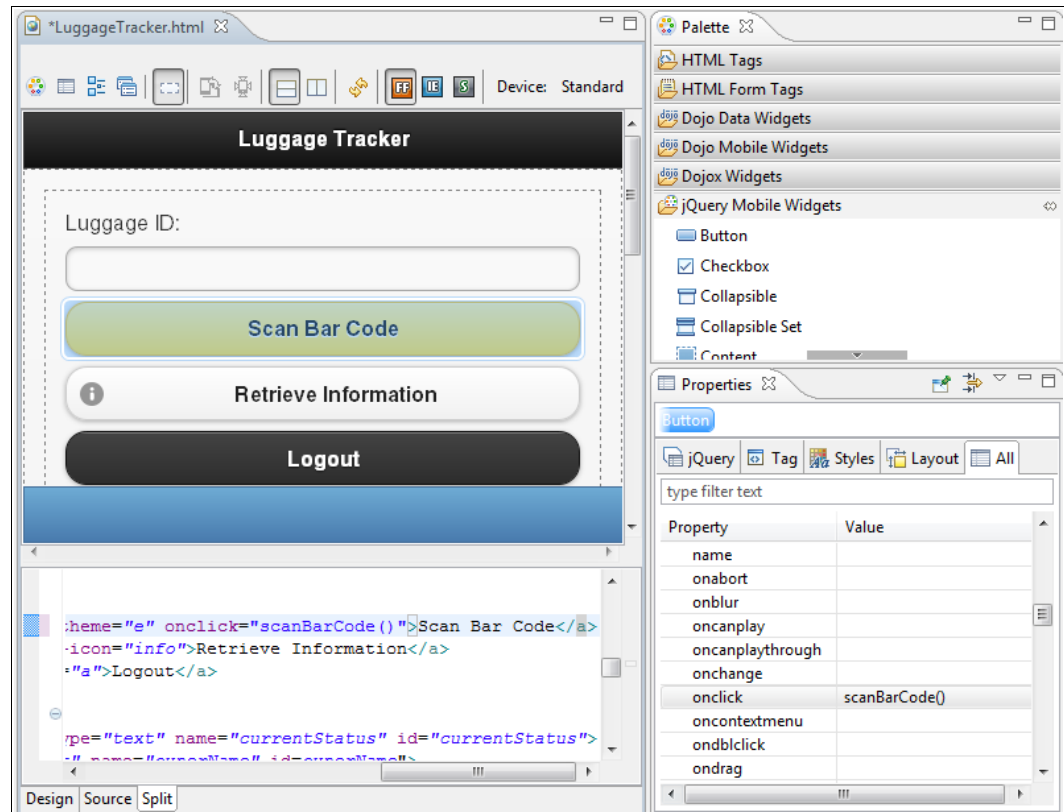


Figure 4-55 Connecting the Scan Bar Code button to the scanBarCode function

5. Save and close the LuggageTracker.html file.

Integrating the client-side authentication

To integrate the client-side authentication procedure created in 4.8.3, “Creating the client-side authentication component” on page 140, use the following steps:

1. In Worklight Studio, expand the CompanyAirlines project until the following folder is visible:
apps/LuggageTracker/common
2. Open the LuggageTracker.html file and go to the Source tab.
3. At the bottom of the file, after the comment `<!--application UI goes here-->`, there are three JavaScript file references. Add a new reference to the `auth.js` file by adding the following line before the `</body>` tag:

```
<script src="js/auth.js"></script>
```
4. Save and close the LuggageTracker.html file.

Integrating the business services adapter

The business services adapter, created in 4.7.3, “Creating the business services adapter” on page 99, is used to retrieve the luggage information from the back-end LTIS and, when necessary, to change the delivery address information in the luggage record. Therefore, the buttons on the user interface must be connected to the functions in the business services adapter using JavaScript functions.

Retrieve Information button functions

When the luggage identifier number is entered (either by scanning the luggage receipt or manually entering it), the CSR clicks the Retrieve Information button to get the current luggage information from the airline's back-end systems. This retrieval is accomplished through the business services adapter, and there are three steps to enable the application to do it:

1. Create the JavaScript function to call the business services adapter when the Retrieve Information button is clicked.
2. Create the JavaScript functions to handle both return cases (success and failure).
3. Connect the Retrieve Information button to the function that calls the adapter.

First, create the JavaScript function to call the business services adapter:

1. Open the `LuggageTracker.js` file in the `apps/LuggageTracker/common/js` folder.
2. Add the JavaScript for the `retrieveInformation` function, shown in Example 4-30, immediately beneath the `scanBarCode` function.

The function verifies that the `luggageID` field contains a value (it is not empty) and if so, invokes the business services adapter's `getLuggageInformation` procedure, passing in the value in the `luggageID` field. For the return, the procedure invokes either the `loadLuggageInformation` method (if the luggage information was successfully retrieved) or the `luggageNotFound` method (if the luggage information was not successfully retrieved because the luggage identifier was not found in the LTIS).

Example 4-30 JavaScript to retrieve the luggage information using the business services adapter

```
function retrieveInformation() {
    var luggageID = $("#LuggageTrackingID").val();
    if ((!luggageID) || (luggageID == '')) {
        document.getElementById("notifications").innerHTML = "Tracking Code can't be empty";
        return;
    }

    var invocationData = {
        adapter : 'BusinessServicesAdapter',
        procedure : 'getLuggageInformation',
        parameters : [ luggageID ]
    };

    WL.Client.invokeProcedure(invocationData, {
        onSuccess : loadLuggageInformation,
        onFailure : luggageNotFound,
    });
}
```

Next, create the JavaScript functions to handle the success and failure return cases:

1. Add the `loadLuggageInformation` function shown in Example 4-31 on page 147 to the end of the `LuggageTracker.js` file.

This function is called when a successful return is received from the adapter procedure call. It populates the input fields in the Luggage information screen with the values returned from the adapter. It then hides the Luggage ID input screen and shows the Luggage information screen.

Example 4-31 JavaScript for the loadLuggageInformation function

```
function loadLuggageInformation(result) {
    var lugInfo = result.invocationResult.luggage;
    document.getElementById("notifications").innerHTML = "";
    WL.Client.logActivity("getLuggageInformation adaptor method invoked
successfully");
    // Luggage tracking info
    $("#l_Status").val(lugInfo.status);
    $("#l_Owner").val(lugInfo.ownerName);
    $("#l_location").val(lugInfo.currentLocation);
    $("#l_NextLocation").val(lugInfo.nextLocation);
    $("#l_FinalLocation").val(lugInfo.finalDestination);
    $("#l_comments").val(lugInfo.comments);

    // Delivery address Info
    $("#l_DeliveryName").val(lugInfo.deliveryAddress.name);
    $("#l_DeliveryAdd1").val(lugInfo.deliveryAddress.addressLine1);
    $("#l_DeliveryAdd2").val(lugInfo.deliveryAddress.addressLine2);
    $("#l_DeliveryCity").val(lugInfo.deliveryAddress.city);
    $("#l_DeliveryState").val(lugInfo.deliveryAddress.state);
    $("#l_DeliveryZip").val(lugInfo.deliveryAddress.zipCode);
    $("#l_DeliveryPhone").val(lugInfo.deliveryAddress.phoneNumber);
    $("#l_DeliveryComments").val(lugInfo.deliveryAddress.comments);

    $("#luggageIdInput").hide();
    $("#l_Information").show();
}
```

2. Add the luggageNotFound function shown in Example 4-32 to the end of the LuggageTracker.js file. This is needed so an error message can be displayed when the luggage adapter does not find the luggage identifier number.

Example 4-32 JavaScript for luggageNotFound function

```
function luggageNotFound(result) {
    WL.Client.logActivity("error:" + result.invocationResult.errors);
    document.getElementById("notifications").innerHTML = "The call to the server
is unsuccessful";
}
```

3. Save the LuggageTracker.js file but do not close it yet.

Finally, connect the Retrieve Information button to the function that calls the adapter:

1. Open the LuggageTracker.html file within the apps/LuggageTracker/common folder.
2. Open the Properties view for the Retrieve Information button either by selecting the button (if the screen is not hidden) or selecting the HTML tag for the button on the Source tab.
3. In the Properties view, click the **All** tab and locate the button's onclick property.
4. Click in the **Value** column of the onclick property and enter retrieveInformation().
5. Save the LuggageTracker.html but do not close it yet.

The CSR can now log in and scan the passenger's luggage receipt (using the bar code scanner component of the application) to retrieve detailed luggage information from the back-end system, which is then shown on the Luggage information screen.

Change Delivery Address button functions

If the passenger's delivery address needs to be updated, the CSR enters the new information, including any additional comments, to be saved in the airline's back-end LTIS. This update is done through the business services adapter, and there are four steps to enable the application to do it:

1. Hide and show the proper screens to the CSR after the Change Delivery Address button is clicked.
2. Create the JavaScript function to gather the address data that is entered and then call the business services adapter when the Save and return button is clicked.
3. Create the JavaScript functions to handle both return cases (success and failure)
4. Connect the Save and return button to the function that calls the adapter

First, hide and show the proper screens to the CSR:

1. Switch to the `LuggageTracker.js` file, which is still open in Rich Page Editor.
2. Append the `updateDeliveryAddress` function, shown in Example 4-33, to the end of the file. This function displays the Luggage delivery address screen and hides the other three screens.

Example 4-33 JavaScript for updateDeliveryAddress function

```
function updateDeliveryAddress() {  
    var lugID = $("#LuggageTrackingID").val();  
    document.getElementById("notifications").innerHTML = "";  
    $("#l_Information").hide();  
    $("#loginForm").hide();  
    $("#luggageIdInput").hide();  
    $("#l_trackingID").val(lugID);  
    $("#setDeliveryInfo").show();  
}
```

3. Save the `LuggageTracker.js` file but do not close it.
4. Switch to the `LuggageTracker.html` file, which is still open in Rich Page Editor.
5. Change the `onclick` property of the Change Delivery Address button to `updateDeliveryAddress()`, using the same process that was used in steps 2 on page 147 through 4 on page 147.
6. Save the `LuggageTracker.html` file but do not close it.

Next, create the JavaScript function to gather the address data entered by the CSR and then call the business services adapter when the Save and return button is clicked:

1. Switch to the `LuggageTracker.js` file, which is still open in Rich Page Editor.
2. Add the JavaScript for the two delivery address-related functions shown in Example 4-34 on page 149 to the end of the file.

The `submitUpdatedDeliveryAddress` function calls the `updateLuggageInformation` function, hides the Luggage delivery address screen, and shows the Luggage information screen. The `updateLuggageInformation` function invokes the business services adapter's `addLuggageDeliveryInformation` procedure to update information in the back-end LTIS.

Example 4-34 JavaScript for submitUpdatedDeliveryAddress, updateLuggageInformation functions

```
function submitUpdatedDeliveryAddress() {
    document.getElementById("notifications").innerHTML = "";
    updateLuggageInformation($("#LuggageTrackingID").val(),
        $("#l_DeliveryName").val(), $("#l_DeliveryAdd1").val(), $(
            "#l_DeliveryAdd2").val(), $("#l_DeliveryCity").val(), $(
            "#l_DeliveryState").val(), $("#l_DeliveryZip").val(), $(
            "#l_DeliveryPhone").val(), $("#l_DeliveryComments").val());

    $("#setDeliveryInfo").hide();
    $("#l_Information").show();
}

function updateLuggageInformation(luggageId, name, addressLine1, addressLine2,
    city, state, zipCode, phoneNumber, comments) {
    var invocationData = {
        adapter : 'BusinessServicesAdapter',
        procedure : 'addLuggageDeliveryInformation',
        parameters : [ luggageId, name, addressLine1, addressLine2, city,
            state, zipCode, phoneNumber, comments ]
    };
    WL.Client.invokeProcedure(invocationData, {
        onSuccess : reloadLuggageInformation,
        onFailure : luggageNotUpdated,
    });
}
```

Next, create the JavaScript functions to handle both the success and failure return cases:

1. Add the two functions shown in Example 4-35 to end of the LuggageTracker.js file.

The reloadLuggageInformation function reloads the fields in the Luggage information screen with the updated address and comments, and displays a message in the notification area to indicate that the update was successful.

The luggageNotUpdated function reloads the old address and comments in the Luggage information screen and displays a message stating that the update failed.

Example 4-35 JavaScript for reloadLuggageInformation and luggageNotUpdated functions

```
function reloadLuggageInformation(result) {
    WL.Client.logActivity("Luggage delivery address updated.");
    var lugInfo = result.invocationResult.luggage;
    $("#l_DeliveryName").val(lugInfo.deliveryAddress.name);
    $("#l_DeliveryAdd1").val(lugInfo.deliveryAddress.addressLine1);
    $("#l_DeliveryAdd2").val(lugInfo.deliveryAddress.addressLine2);
    $("#l_DeliveryCity").val(lugInfo.deliveryAddress.city);
    $("#l_DeliveryState").val(lugInfo.deliveryAddress.state);
    $("#l_DeliveryZip").val(lugInfo.deliveryAddress.zipCode);
    $("#l_DeliveryPhone").val(lugInfo.deliveryAddress.phoneNumber);
    $("#l_DeliveryComments").val(lugInfo.deliveryAddress.comments);
    document.getElementById("notifications").innerHTML = "Address updated
successfully";
}

function luggageNotUpdated(result) {
    var lugInfo = result.invocationResult.luggage;
    WL.Client.logActivity("Failed to update luggage delivery address.");
}
```

```

$("#1_DeliveryName").val(lugInfo.deliveryAddress.name);
$("#1_DeliveryAdd1").val(lugInfo.deliveryAddress.addressLine1);
$("#1_DeliveryAdd2").val(lugInfo.deliveryAddress.addressLine2);
$("#1_DeliveryCity").val(lugInfo.deliveryAddress.city);
$("#1_DeliveryState").val(lugInfo.deliveryAddress.state);
$("#1_DeliveryZip").val(lugInfo.deliveryAddress.zipCode);
$("#1_DeliveryPhone").val(lugInfo.deliveryAddress.phoneNumber);
$("#1_DeliveryComments").val(lugInfo.deliveryAddress.comments);
document.getElementById("notifications").innerHTML = "The call to the server
is unsuccessful";
}

```

2. Save and close the LuggageTracker.js file.

Finally, connect the Save and return button to the submitUpdatedDeliveryAddress function. This process involves modifying the LuggageTracker.html file, which is the last file open in Rich Page Editor. If it is not visible, switch to it and then use the following steps:

3. Open the Properties view for the Save and return button, either by selecting the button (if the screen is not hidden) or by selecting the HTML tag for the button on the Source tab.
4. In the Properties view, click the **All** tab and locate the button's onclick property.
5. Click in the **Value** column of the onclick property and enter `submitUpdatedDeliveryAddress()`.
6. Save and close the LuggageTracker.html file.

The buttons on the user interface are now connected to the JavaScript functions needed to complete the various user stories. The mobile application is now ready to be unit tested by the developer using IBM Worklight Studio.

4.8.5 Unit testing the application

As the mobile application is developed, the mobile application developer does unit testing on each part of the application to validate the components individually. This section describes the steps that are involved in unit testing using device simulators, device emulators, and physical devices.

A web application, which is an application that contains only HTML pages and functions, can be unit tested with just a *mobile browser simulator*. Hybrid and native applications must be unit tested using *mobile device emulators* (provided with the platform-specific SDKs that you downloaded) that can mimic actual mobile devices for testing shell components and native modules.

This means that for LuggageTracker, mobile browser simulators can be used to test all application functions except the bar code scanner component, which must be tested using a mobile device emulator.

After testing the various application components with simulators or emulators, the next level of unit testing uses an actual device connected to the workstation of the mobile application developer. This step typically involves debugging errors by monitoring the information presented on the SDK console.

Creating the environments for testing

Before unit testing can begin, a Worklight environment must be created for the target platform. These environments are used to generate the application binaries for the mobile platforms.

To create the Worklight environments for testing LuggageTracker, use the following steps:

1. Start IBM Worklight Studio if it is not already started.
2. Expand the CompanyAirlines project in Project Explorer until the LuggageTracker application folder is visible.
3. Right-click the LuggageTracker folder and select **New** → **Worklight Environment**.
4. In the New Worklight Environment window, select **iPhone** and **Android phones and tablets**, as shown in Figure 4-56, and then click **Finish**.

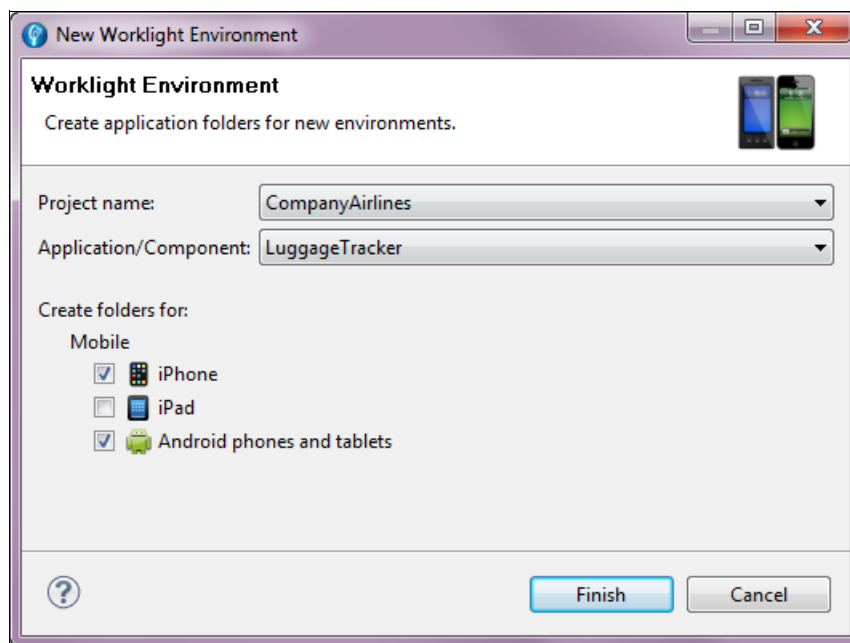


Figure 4-56 Creating the environments for unit testing

Two environments have now been created under the LuggageTracker application folder, as shown in Figure 4-57.

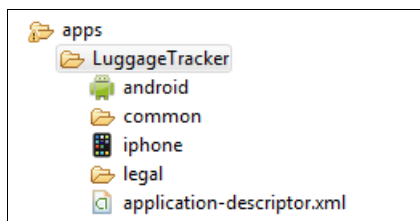


Figure 4-57 LuggageTracker environments shown in Project Explorer

Testing web applications with a mobile browser simulator

IBM Worklight Studio provides device-specific mobile browser simulators for unit testing the web applications and the web technology-based portions of hybrid applications.

Launching the application in the simulator

Start by confirming that the back-end services to be used in your test environment have sufficient luggage identifier data for testing, then use the following steps:

1. Start IBM Worklight Studio, if it is not already started.
2. Expand the CompanyAirlines project in Project Explorer until the environment named **android** within the LuggageTracker application folder is visible.
3. Right-click the environment name and then click **Run** → **3 Preview**, as shown in Figure 4-58.

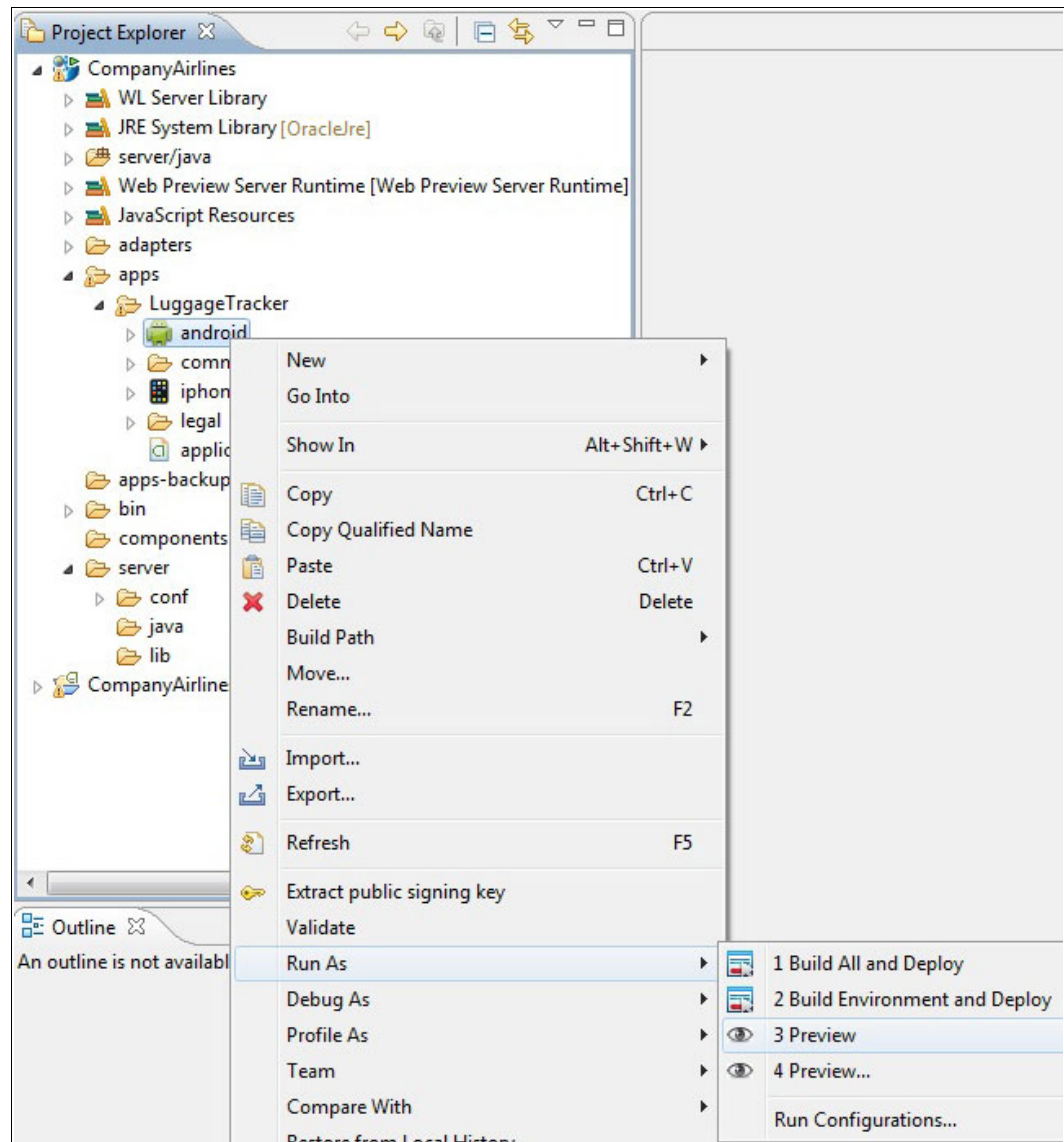


Figure 4-58 Steps to run the Android application in browser simulator

This starts Worklight Studio's mobile browser simulator for Android and runs the LuggageTracker application within it, as shown in Figure 4-59.

HINT: If the mobile application does not display properly in the simulator, set the simulator to act as an external web browser by selecting **Window** → **Preferences** and then choosing **Use external web browser** (located under General → Web Browser).

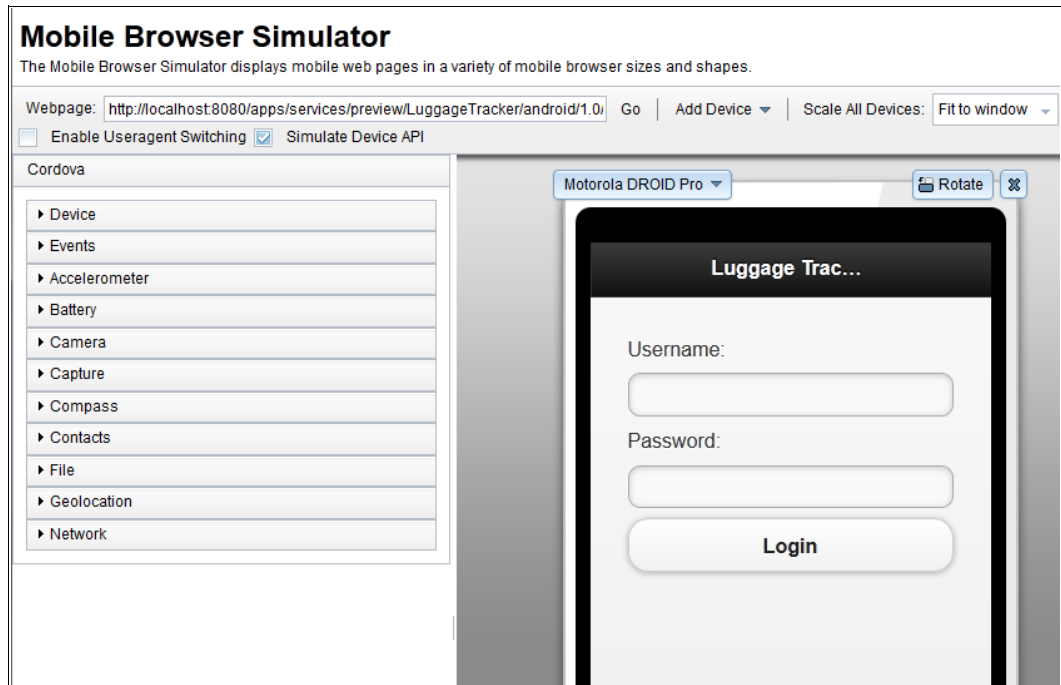


Figure 4-59 LuggageTracker running in browser simulator

The LuggageTracker Login screen is displayed in the simulator and is ready for testing.

Using the mobile browser simulator

With the mobile browser simulator, you can see and interact with the web content of your application. A few of the useful functions of the simulator include the following items:

- ▶ You can choose a different mobile device for simulation by selecting it from the device name box (the blue box that shows the currently selected device, which in Figure 4-59 on page 153 is Motorola DROID Pro).
- ▶ You can rotate the device display by using the **Rotate** button (the blue box shown at the upper-right corner of the simulated device).
- ▶ You can simulate the application on different devices within the same platform family side-by-side, such as to compare the application on different screen sizes. To do this, select a device from the Add Device drop-down list at the top of the page (see Figure 4-60).

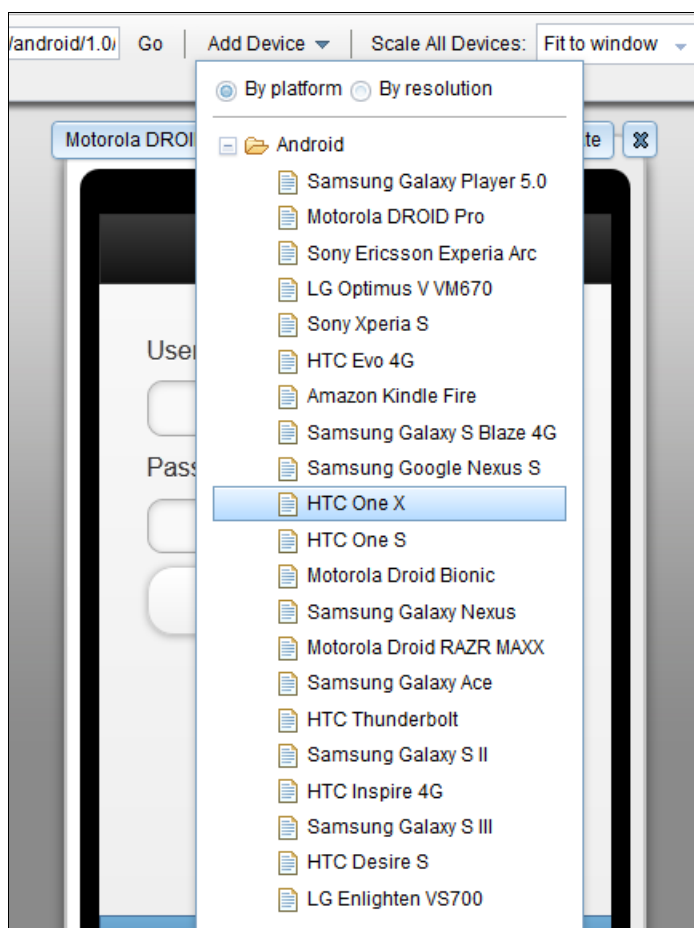


Figure 4-60 Selecting different devices for the simulator

Running the application user scenarios

At this point, the application is running within the browser simulator and you are ready to test the scenarios outlined in the user stories described in 4.1.2, “Agile breakdown and design” on page 62.

To start testing, use the following steps:

1. On the login screen, enter a valid user name and password and then click **Login**. The Luggage ID input form is then displayed, as shown in Figure 4-61.

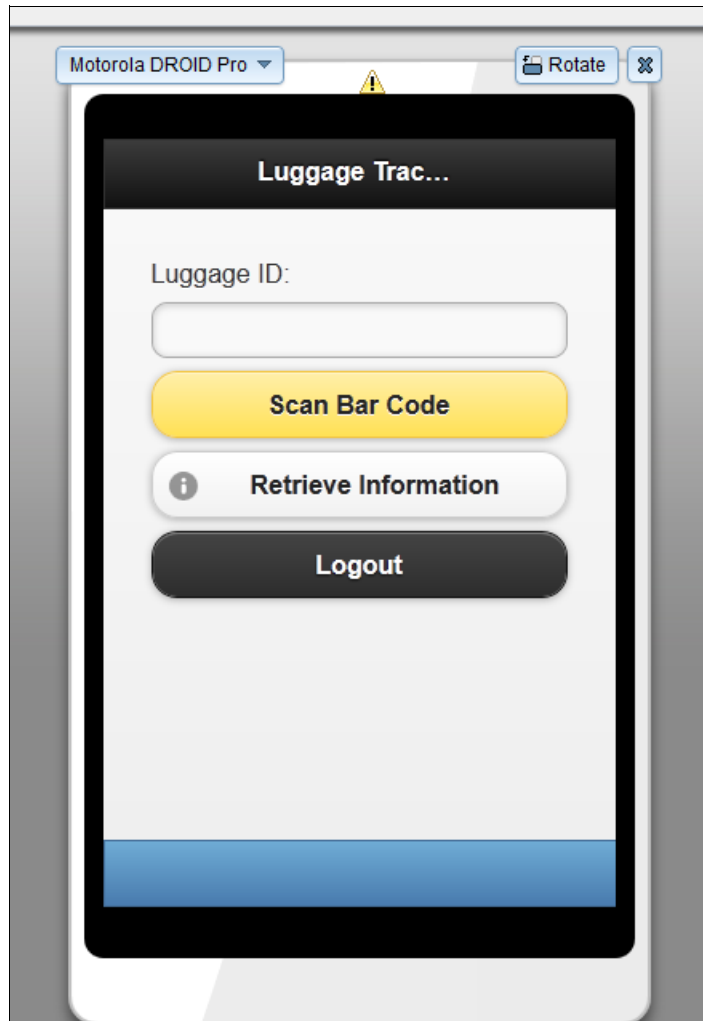


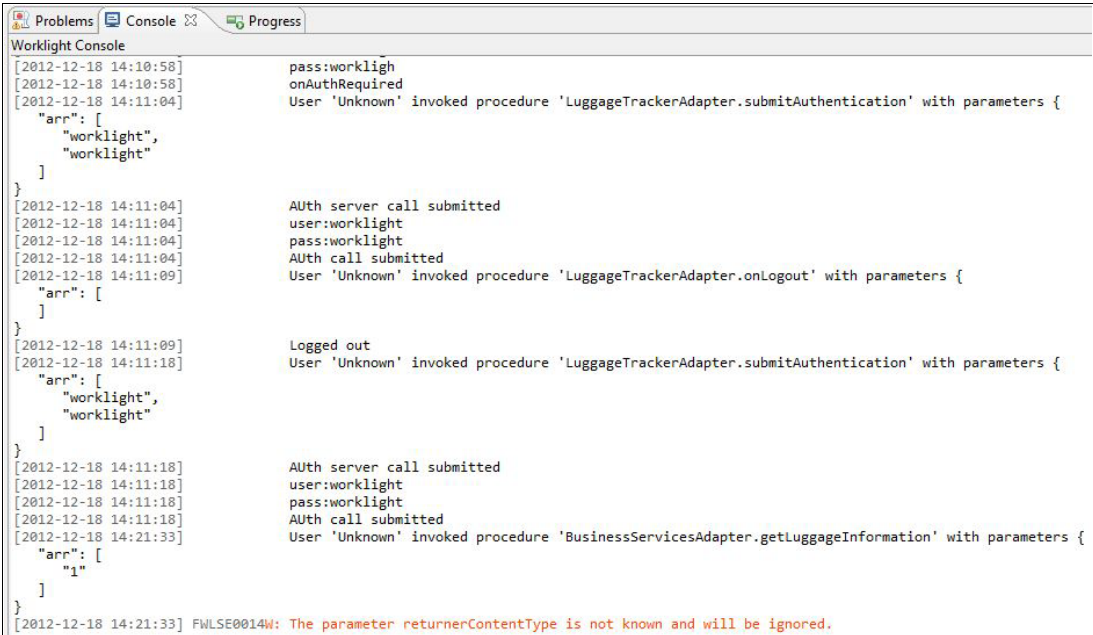
Figure 4-61 Luggage ID input screen displayed with a mobile browser simulator

2. Because the browser simulator has no native device functionality, and the bar code scanner component is a native function, you cannot simulate the scanning operation and instead must manually enter a valid luggage identifier number into the text field and then click **Retrieve Information**. If this is successful, the Luggage information screen is displayed.
3. Next, test the delivery address functionality by clicking **Change Delivery Address**. This displays the Luggage delivery address screen.
4. Enter a test delivery address into the available fields and then click **Save and return**. The address information is sent to the BusinessServicesAdapter to be saved in the simulated

back-end systems being used for testing, after which the Luggage information screen is displayed again.

5. Click **Close** to leave the luggage information portion of the application and return to the Luggage ID input screen.
6. Repeat these steps with different luggage identifier numbers and delivery addresses and then retrieve information for the records you updated to verify that the new information was recorded properly.
7. Click **Logout** to close the application and return to the Login screen.

If problems are encountered, you can view debugging information in the Console view of IBM Worklight Studio, shown in Figure 4-62. Worklight Logger calls using the `WL.Logger.debug` method will be shown in the Console and can be used to debug the flow through certain JavaScript functions and display the values of variables.



```
Worklight Console
[2012-12-18 14:10:58] pass:worklight
[2012-12-18 14:10:58] onAuthRequired
[2012-12-18 14:11:04] User 'Unknown' invoked procedure 'LuggageTrackerAdapter.submitAuthentication' with parameters {
  "arr": [
    "worklight",
    "worklight"
  ]
}
[2012-12-18 14:11:04] Auth server call submitted
[2012-12-18 14:11:04] user:worklight
[2012-12-18 14:11:04] pass:worklight
[2012-12-18 14:11:04] Auth call submitted
[2012-12-18 14:11:09] User 'Unknown' invoked procedure 'LuggageTrackerAdapter.onLogout' with parameters {
  "arr": [
  ]
}
[2012-12-18 14:11:09] Logged out
[2012-12-18 14:11:18] User 'Unknown' invoked procedure 'LuggageTrackerAdapter.submitAuthentication' with parameters {
  "arr": [
    "worklight",
    "worklight"
  ]
}
[2012-12-18 14:11:18] Auth server call submitted
[2012-12-18 14:11:18] user:worklight
[2012-12-18 14:11:18] pass:worklight
[2012-12-18 14:11:18] Auth call submitted
[2012-12-18 14:21:33] User 'Unknown' invoked procedure 'BusinessServicesAdapter.getLuggageInformation' with parameters {
  "arr": [
    "1"
  ]
}
[2012-12-18 14:21:33] FWLSE0014W: The parameter returnerContentType is not known and will be ignored.
```

Figure 4-62 Showing debug statements using the Worklight Studio Console view

A suggested approach is to perform the same steps on several simulated devices with different screen sizes. This approach can help you verify that the application displays and functions properly on various screen sizes and resolutions.

After completing your simulator testing, you can test the native functions of the application using a native device emulator. Then you can move on to testing the application with physical devices to evaluate the send user experience.

Testing hybrid and native applications with device emulators

To test shell components or native API code, developers must use the mobile device emulators that are included in the platform-specific SDKs that were downloaded previously. An Android Virtual Device (AVD) was created when you completed the steps detailed in 4.2.1, “Setting up the development environment” on page 68 (the actual installation instructions for the SDK are in 6.3.2, “Installing the Android SDK on Mac OS X” on page 303). This AVD provides a way to test the native code components and application user interface using the same physical device environment but without the actual device.

Follow these steps to test native application features on your Android virtual device:

1. Start IBM Worklight Studio if it is not already started.
2. Expand the CompanyAirlines project in Project Explorer until the environment named android (located under the LuggageTracker application folder) is visible.
3. Right-click the environment named android and select **Run As** → **1 Build All and Deploy**, as shown in Figure 4-63. This builds the Android platform environment and creates a new project named CompanyAirLinesLuggageTrackerAndroid.

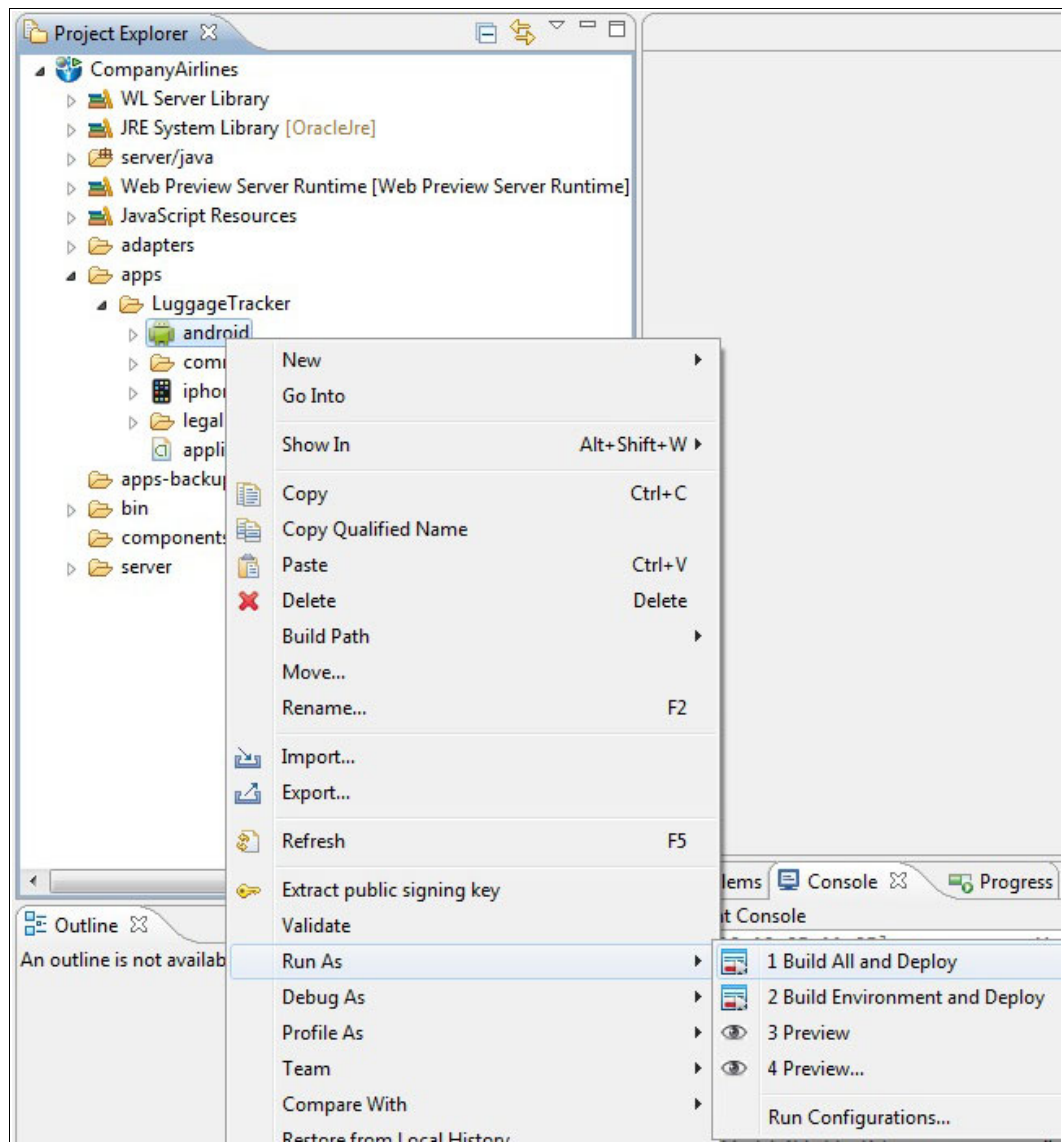


Figure 4-63 Building the application for the Android virtual device

4. When the environment build is complete, switch to the Java perspective by selecting **Window** → **Open Perspective** → **Other** and then selecting **Java**.
5. In Project Explorer, right-click the newly created Java project, CompanyAirLinesLuggageTrackerAndroid, and then select **Run As** → **1 Android Application**, as shown in Figure 4-64 on page 158.

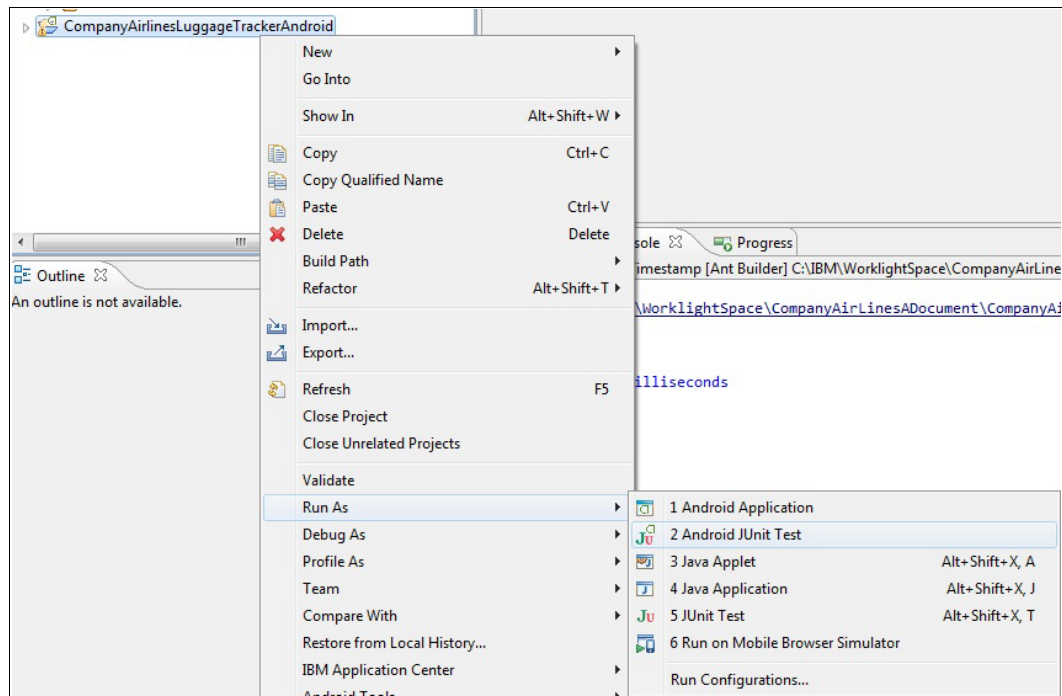


Figure 4-64 Deploying the application to the virtual Android device

The application is now running on the mobile device emulator, as shown in Figure 4-65, and is ready for testing.

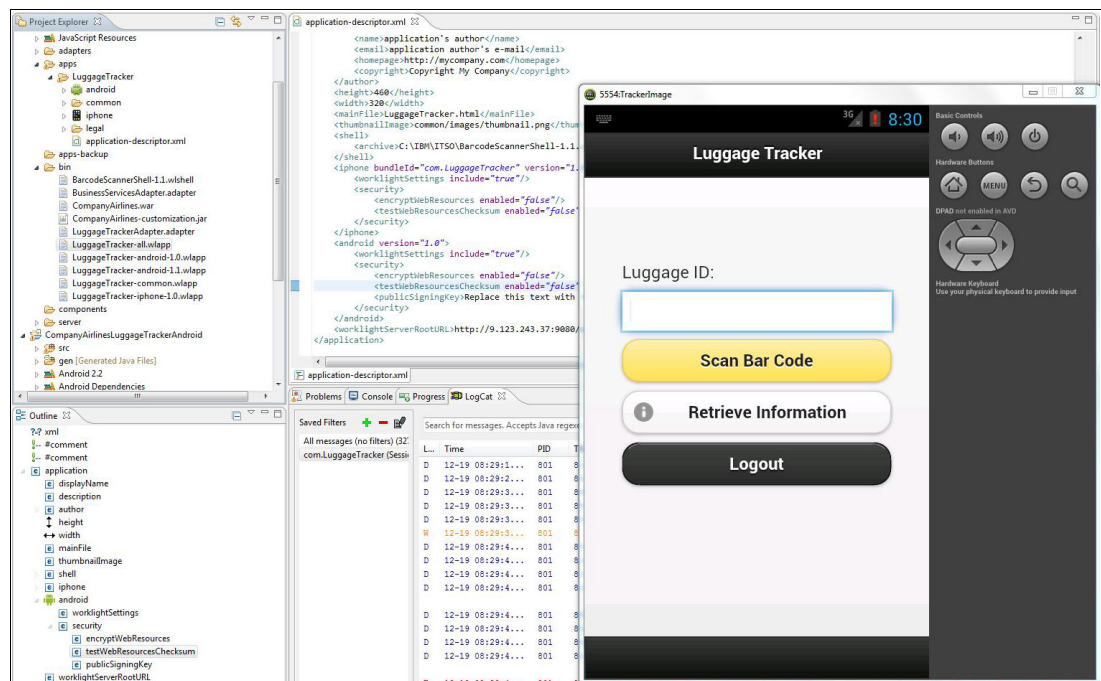


Figure 4-65 LuggageTracker running in mobile device emulator for Android

Complete your testing of the application functions. After the testing is complete, the next step is to test the application on physical devices.

Testing by using physical devices

Unit testing on physical devices is critical for application debugging, because it is the only way to evaluate the actual user experience. To do this, start by connecting a mobile device to the developer's workstation using a USB cable. Ensure that all device drivers are installed and working properly before you deploy the application to the mobile device.

To test the application on the device you connected to the workstation, repeat the steps in "Testing hybrid and native applications with device emulators" on page 156. After clicking **Run as** → **Android Application**, the device manager detects that a physical device is connected and prompts you to indicate whether to use the device emulator or the connected device, as shown in Figure 4-66. Select the option for the connected device and then click **OK**.

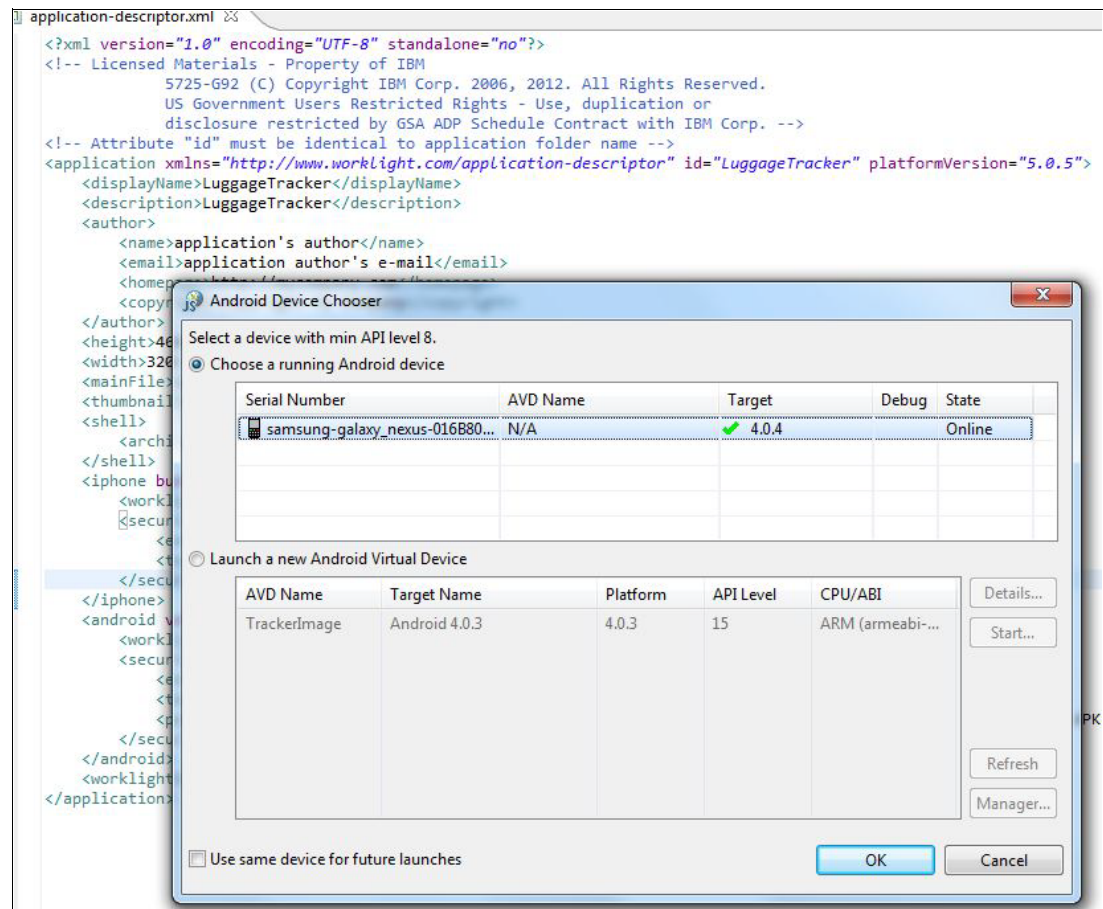


Figure 4-66 Selecting which device to deploy the application to

The Android SDK Manager deploys and launches the application on the selected physical device so you can test it as shown in Figure 4-67. As with using the emulator, you can use the Console view in Worklight Studio to assist in debugging any application errors that are found.

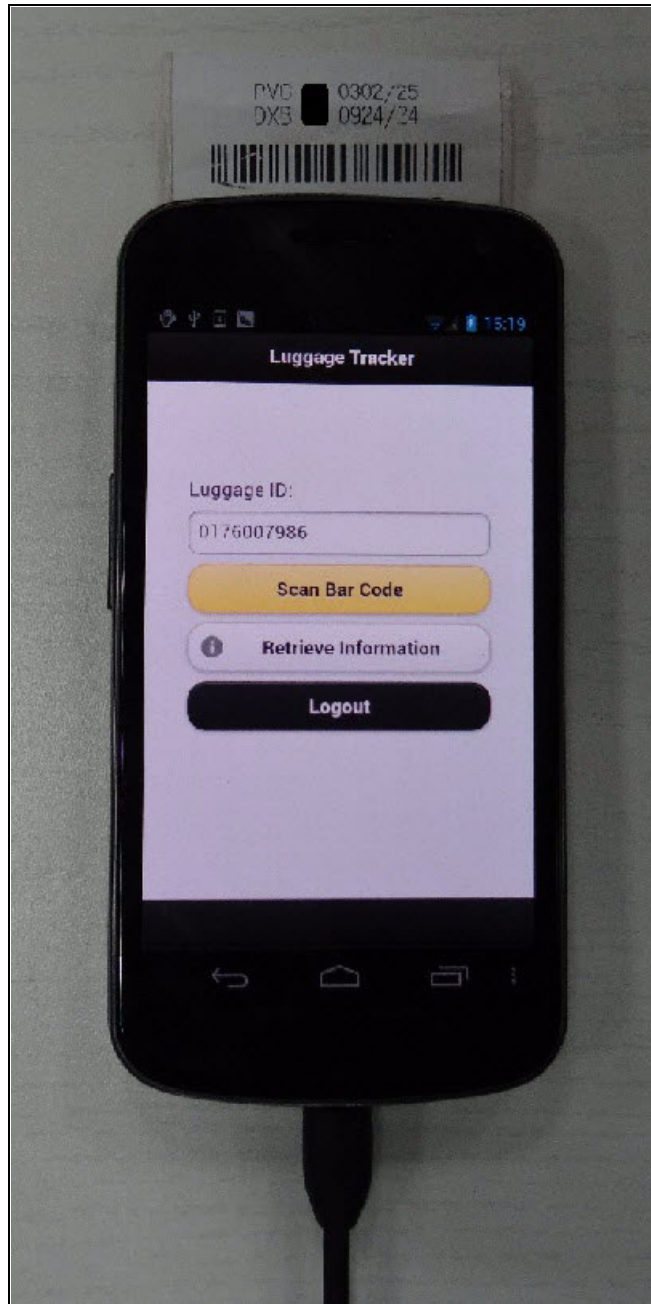


Figure 4-67 LuggageTracker running on USB-connected physical device

4.8.6 Building the mobile application

At this point, the major components of the mobile application are created and unit tested:

- ▶ User interface
- ▶ Application business logic for data manipulations
- ▶ Shell components for native APIs and libraries
- ▶ Security features for user authentication

The next step is to integrate all of the components to create the fully functioning mobile application. After that, the application is readied for deployment to the mobile testers for end-to-end testing, and then it will be released for use in the field by the airline's CSRs.

To build the final application, the following files must be generated:

- ▶ Worklight adapters from the AirlineAdapterProject project (multiple .adapter files)
- ▶ Worklight web application (.war file)
- ▶ Worklight mobile application (multiple .wlapp files)
- ▶ Native Android application (.apk file)
- ▶ Native iOS application (.ipa file)

Before you begin, confirm that you imported all of the projects that were previously created in your Worklight Studio workspace. The instructions in this book put all of these projects in the same workspace, but in a real-life development effort, the components are created by different developers in different roles throughout the IT organization.

The following projects that are present in your Worklight Studio workspace:

- ▶ CompanyAirlines: The user interface
- ▶ AirlineAdapterProject: The adapters

Within an enterprise, there are multiple target environments, such as development, quality assurance, pre-production, and production. For this book, development and pre-production are the only target environments used. The development environment for Company Airline A's mobile development team is Worklight Studio. The pre-production environment was described in 4.2.2, "Setting up the pre-production environment" on page 69.

To build the mobile application for a specific target environment, several tasks are required:

- ▶ Verifying the servers and ports
- ▶ Setting the application context root
- ▶ Building the adapters
- ▶ Building the application files
- ▶ Exporting the Android application
- ▶ Exporting and signing the iOS application

Verifying servers and ports for the target environment

Before you can start bringing together the application components, you must verify that all of the server addresses match the environment on which the mobile application will be deployed. For example, the pre-production environment normally replicates the production environment, although the server names are probably different.

Use the following steps to edit the necessary files to ensure that the server names and port numbers match targeted deployment environment:

1. Open the BusinessServicesAdapter.xml file in the adapters/BusinessServicesAdapter folder of the AirlineAdapterProject project, as shown in Figure 4-68 on page 162.

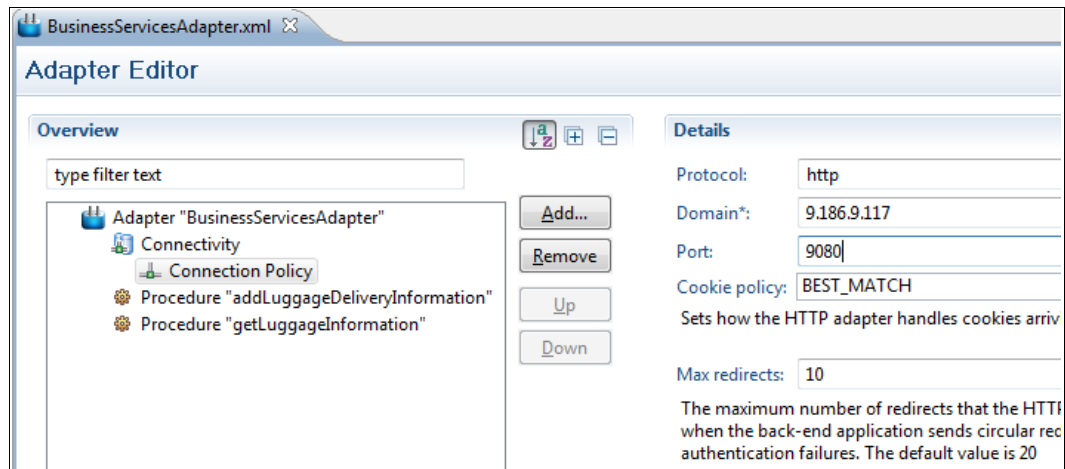


Figure 4-68 Verifying the connection information for the adapter

2. Verify that the Connection Policy details in the file (protocol, domain, and port) match the details for the back-end services provider in the target environment.
3. Save and close the BusinessServicesAdapter.xml file.
4. Open the LuggageTrackerAdapter.xml file in the adapters/LuggageTrackerAdapter folder of the AirlineAdapterProject project.
5. Verify that the Connection Policy details in this file (protocol, domain, and port) match the details for the authentication provider in the target environment.
6. Save and close the LuggageTrackerAdapter.xml file.

Setting application context root for the target environment

Next, the application descriptor file must be modified to use the host name or IP address of the Worklight Server. In most cases, the development team will use the local development server provided by Worklight Studio; the pre-production and production environments will have their own Worklight Server. To ensure that the mobile application is configured with the correct Worklight Server URL, use the following steps:

1. Open the application-descriptor.xml file within the apps/LuggageTracker folder of the CompanyAirlines project.
2. Switch to the Source tab.
3. Locate the line that contains the <worklightServerRootURL> tag.
4. Change the value within the tag to match the target environment using the following format, where <wl_server_name> is the server name of your IBM Worklight Server:

`http://<wl_server_name>:9080/luggagetracker`

In this example, luggagetracker is the context root of the LuggageTracker application for IBM Worklight Server.

5. Save and close the application-descriptor.xml file.

Building the adapters

The LuggageTracker application uses adapters to connect the mobile application to the company's back-end business services and to implement the server-side authentication model.

The following adapters were created and tested by a mobile server developer in 4.7, “Creating the adapters” on page 88:

- ▶ A business services adapter named `BusinessServicesAdapter`
- ▶ An authentication adapter named `LuggageTrackerAdapter`

Now, you can build the adapters to generate the files that will be used to deploy the adapters to Worklight Server:

1. Expand the `AirlineAdapterProject` project and then expand the adapters folder.
2. Right-click the `BusinessServicesAdapter` folder and select **Run As** → **Deploy Worklight Adapter**. This builds the adapters to generate the deployable adapter file, `BusinessServicesAdapter.adapter` file. It also deploys this adapter to the built-in Worklight Server within Worklight Studio.
3. Repeat Step 2 for the `LuggageTrackerAdapter` adapter in the `LuggageTrackerAdapter` folder.

The two adapter files are now located in the `bin` folder of `AirlineAdapterProject` and will be used to deploy the adapters to Worklight Server in the target environment:

- ▶ `BusinessServicesAdapter.adapter`
- ▶ `LuggageTrackerAdapter.adapter`

Building the web application archive and Worklight application files

The project-specific web application archive and Worklight application files must be deployed to Worklight Server in the target environment. For the `LuggageTracker` application, the following files must be built so that they can be deployed to Worklight Server:

- ▶ `CompanyAirlines.war`
- ▶ `LuggageTracker-common.wlapp`
- ▶ `LuggageTracker-android.wlapp`
- ▶ `LuggageTracker-iphone.wlapp`

The project-specific web application archive (`CompanyAirlines.war`) contains configuration data for the project and is named based on the name of the project in Worklight Studio. Examples of the configuration data in this archive include the following items:

- ▶ Security realm definition
- ▶ Application context root

The Worklight application files (the three `.wlapp` files) contain the actual web content of the application, which includes the HTML file that was created when the widgets were laid out in Rich Page Editor and the JavaScript files that were created in the previous sections. Although these files also exist in the native applications (the `.ipa` and `.apk` files), these Worklight application files are used to update the web content without a new version of the native application. For example, the text of a label can be changed or an error in one of the JavaScript files can be fixed by building and deploying a new version of the Worklight application file. The change can then be made to the users device the next time the application connected to Worklight Server using the Direct Update capabilities of IBM Worklight.

To build these files, use the following steps:

1. Expand the `CompanyAirlines` project and then expand the `apps` folder.
2. Right-click the `LuggageTracker` folder and select **Run As** → **Build All and Deploy**.

The web application archive and Worklight application files are now built and are located in the `bin` folder of the `CompanyAirlines` project. The source code for the Android application

has been generated, and a new project, CompanyAirlinesLuggageTrackerAndroid, has been created which is now visible in Project Explorer. This project contains the source code that is required to build the Android binary file (the .apk file) that is described in the next section.

Exporting the Android application

To export the Android application, the Android Development Tools (ADT) plug-in for Eclipse must be installed. You installed these tools as part of the development environment setup for the system administrator in 4.2.1, “Setting up the development environment” on page 68. The export process includes building the .apk file and signing the Android application.

Use the following steps to export the Android application (including generating and signing the .apk file):

1. Right-click the CompanyAirlinesLuggageTrackerAndroid project folder and then select **Android Tools** → **Export Signed Application Package**.
2. The Export Android Application wizard opens. On the first page of the wizard, the Project field is pre-filled, as shown in Figure 4-69. Confirm that the listed project is the correct one and then click **Next**. If it is not the correct project, click **Browse** to find the correct one.

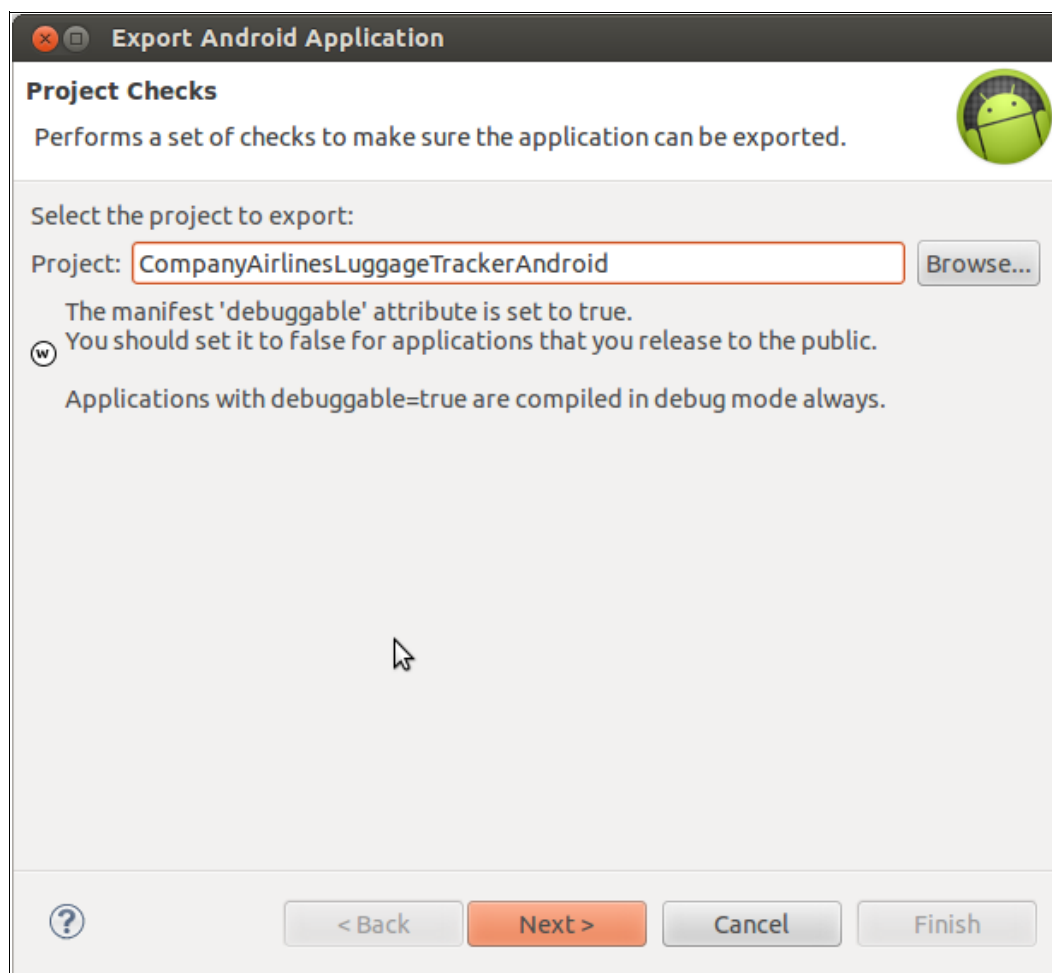


Figure 4-69 Pre-filled project field in the Export Android Application dialog

3. The Keystore selection page of the wizard is displayed (Figure 4-70 on page 165) so you can provide a keystore (new or existing) for the application signing certificate. This certificate is used to sign the Android application (the .apk file). For purposes of this book,

select **Create new keystore** and then click **Browse** to designate a location to save the new keystore.

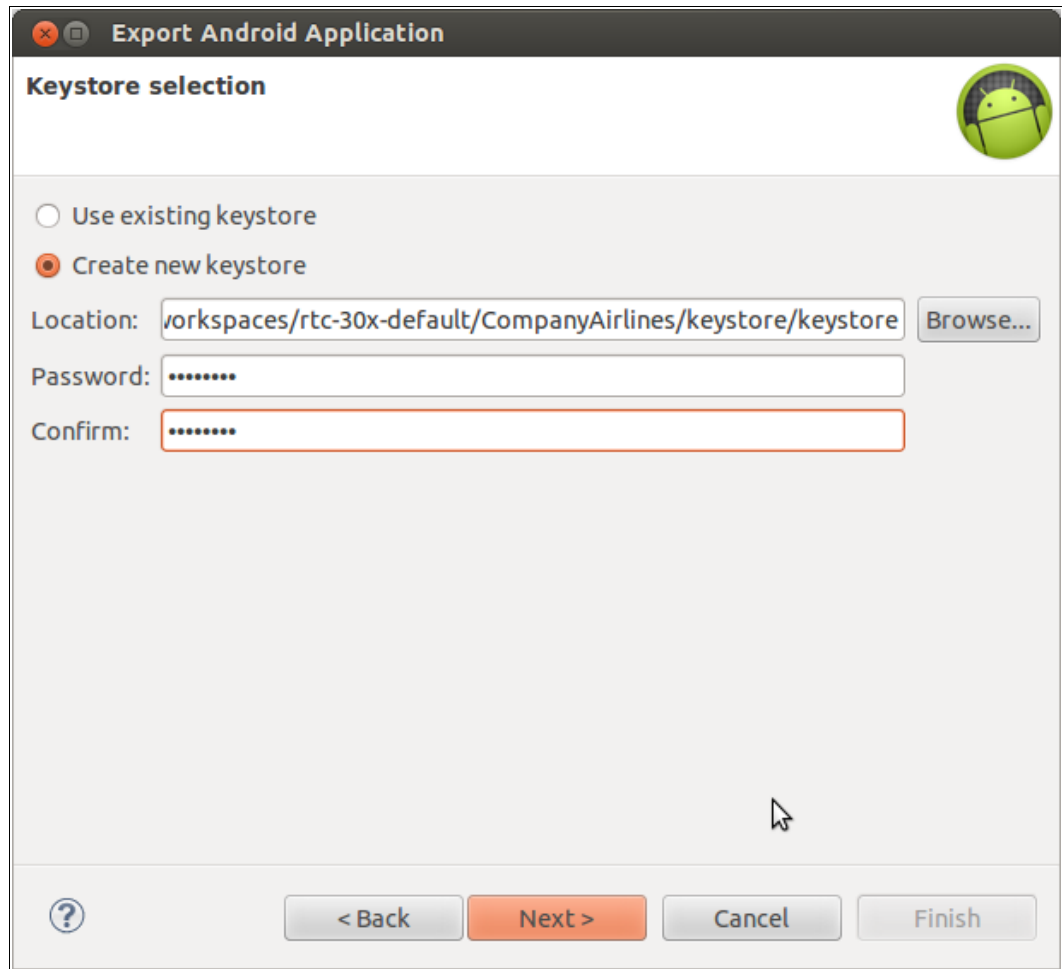
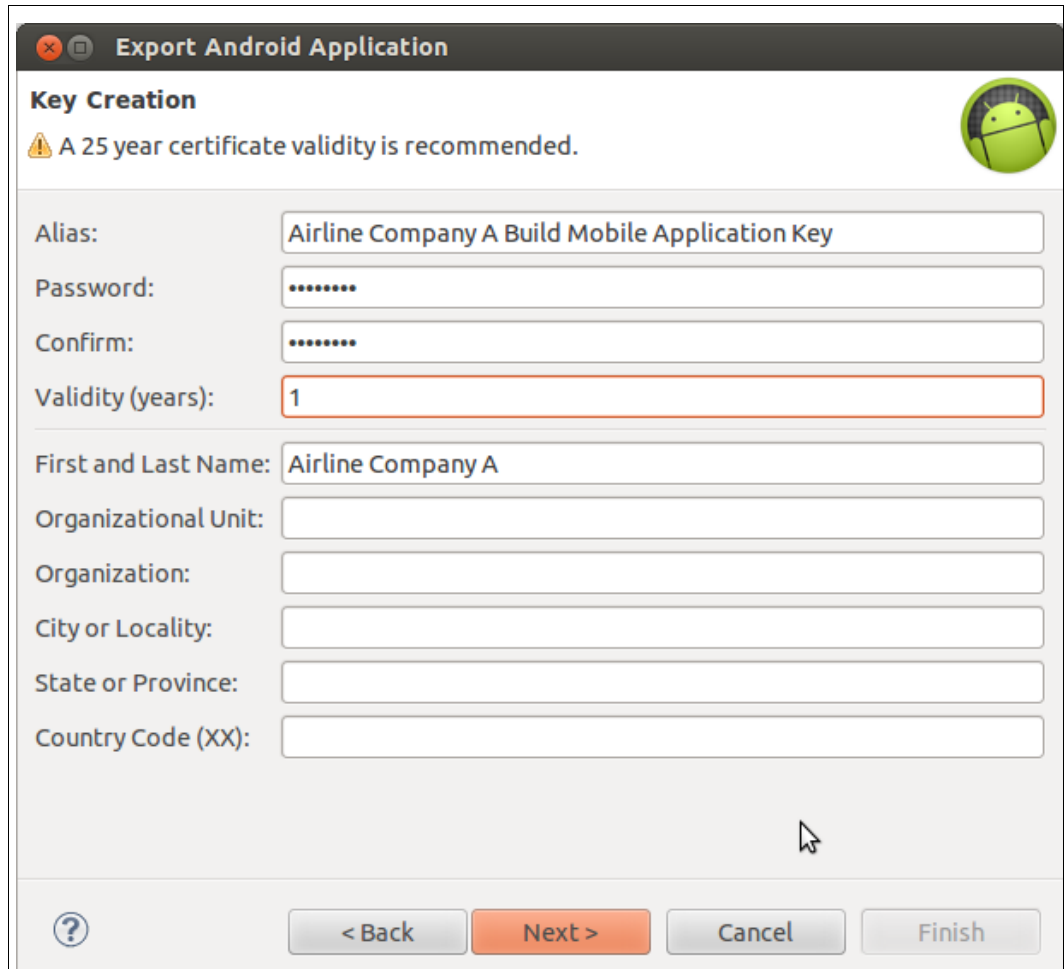


Figure 4-70 Creating a new keystore

4. Enter the password (in this example, it is password) in both the Password and Confirm fields, and then click **Next**.

5. The Key Creation page is displayed (Figure 4-71). This is where you provide the details for creation of the key, including the key alias, the key password (from the previous step), a validity period for the certificate, and the organization details. Fill in this information and then click **Next**.



Export Android Application

Key Creation

⚠ A 25 year certificate validity is recommended.

Alias: Airline Company A Build Mobile Application Key

Password:

Confirm:

Validity (years): 1

First and Last Name: Airline Company A

Organizational Unit:

Organization:

City or Locality:

State or Province:

Country Code (XX):

? < Back Next > Cancel Finish

Figure 4-71 Providing the key creation details

6. The final page of the wizard, Destination and key/certificate checks, is now displayed. In the Destination APK file field, designate a location to store the signed Android application. For purposes of this book, click **Browse** and then select the file named `CompanyAirlinesLuggageTrackerAndroid.apk` in the `bin` folder in the `CompanyAirlines` project, as shown in Figure 4-72.

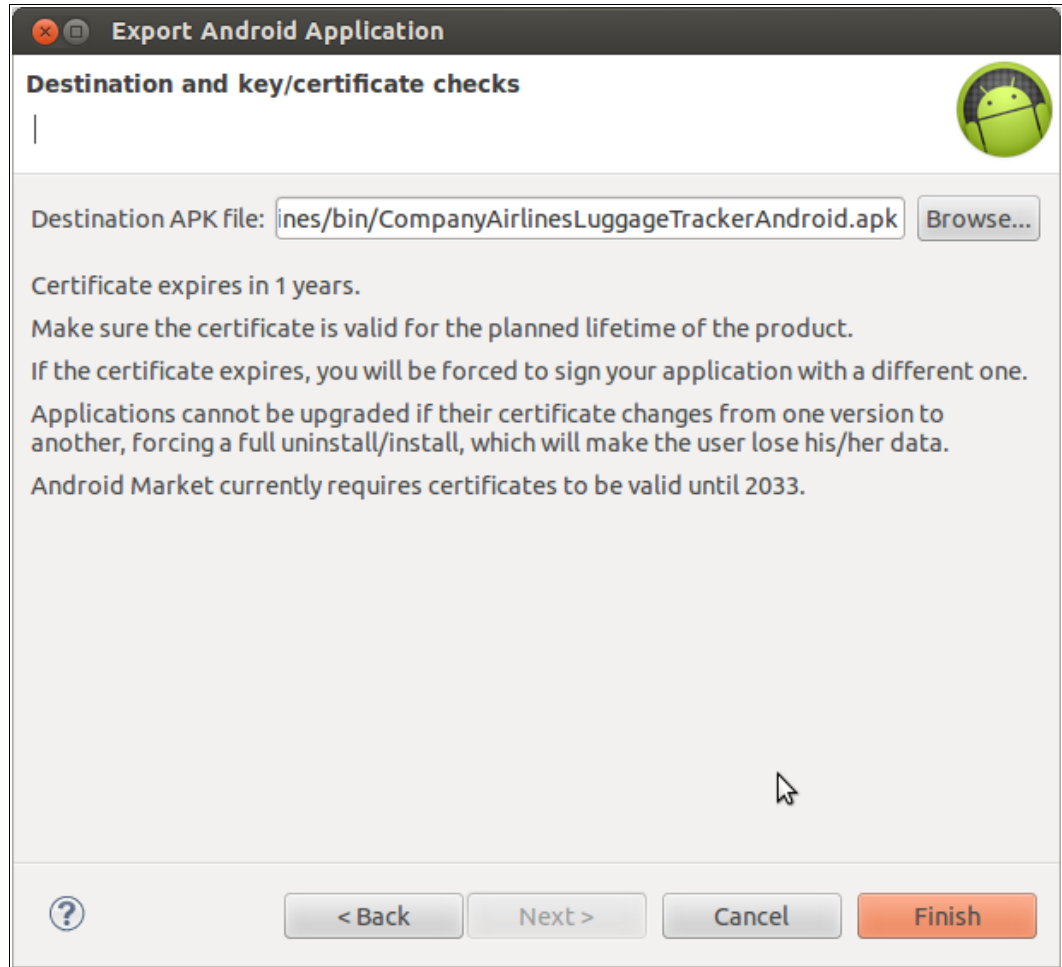


Figure 4-72 Selecting the destination folder for the signed Android application

7. Click **Finish**.

The signed Android mobile application is now in the `bin` folder.

Building and signing the iOS application

As you did for the Android application, build and sign the iOS application to make it available to users. However, unlike the process for the Android application, to do this for the iOS application you need both the device-specific SDK and an account in one of the Apple iOS Developer Programs, as detailed at the Apple Developer website:

<https://developer.apple.com/programs/start/ios/>

Before beginning, ensure that your developer certificate and provisioning profile (obtained by signing up for one of the Apple iOS Developer Programs) are valid and installed on the Mac OS X system on which you will be building and signing the iOS application. Details about this process is in the *iOS Team Administration Guide*:

<http://developer.apple.com/library/ios/#documentation/ToolsLanguages/Conceptual/DevPortalGuide/Introduction/Introduction.html>

Use the following steps to build and sign the iOS application (the .ipa file):

1. Start IBM Worklight Studio if it is not already started.
2. Expand the CompanyAirlines project, right-click the apps/LuggageTracker/iphone directory, and then select **Xcode**. The Xcode IDE will now come to the foreground. If it does not, switch to the Xcode IDE manually.
3. Confirm that the correct developer certificate, obtained when you signed up for one of the Apple iOS Developer Programs, is used to sign the application:
 - a. In the navigation area on the left side of the screen, select the CompanyAirlinesLuggageTrackerIphone project. This displays the Xcode project information in the main section of the IDE.
 - b. In the first column of the main section of the IDE, you see PROJECT and TARGETS headings. The target in Xcode is used to build various applications from the same project, such as a paid version and a free version that share most of the same classes. Select the target CompanyAirlinesLuggageTrackerIphone, which is the only available target, and then choose **Build Settings** from the IDE menu bar.
 - c. Locate the Code Signing Identity setting and make sure it matches your developer certificate, as shown in Figure 4-73.

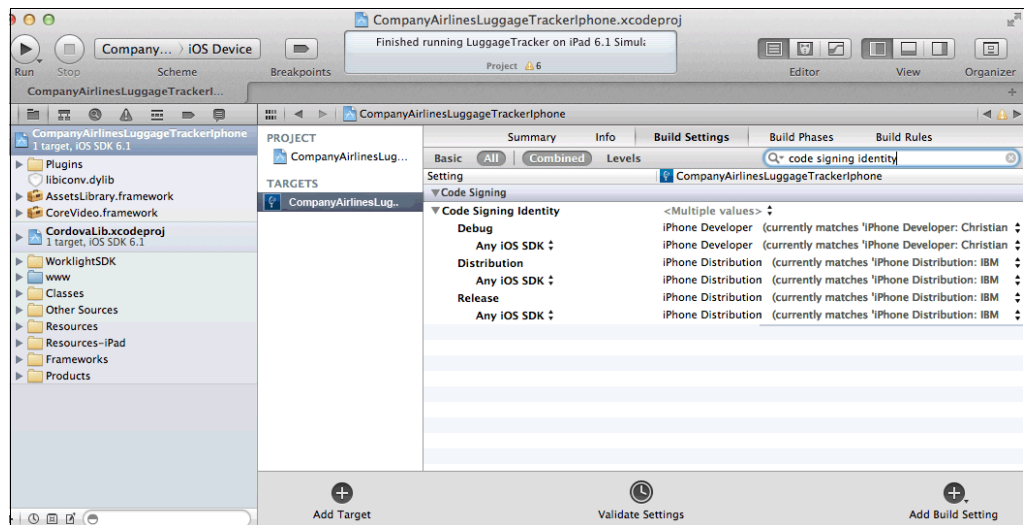


Figure 4-73 Setting code signing identity in Xcode IDE

4. In the upper-left corner of the Xcode IDE, in the main toolbar beside the Run button, select **iOS device**, as shown in Figure 4-74.

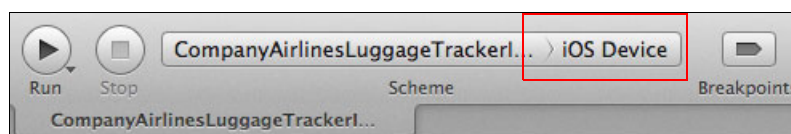


Figure 4-74 Selecting iOS Device as the target for the iPhone version

5. In the Xcode menu, select **Product** → **Archive**. This step builds the application and opens the Organizer (shown in Figure 4-75) with the LuggageTracker application information displayed for the iPhone-specific application.

Troubleshooting: If the Archive option is not available in the menu, make sure that you did not select a simulator in Step 4.

6. In the list of builds at the bottom of the Organizer, select the archive that you just created (look for the creation date in the table), and then click **Distribute**.

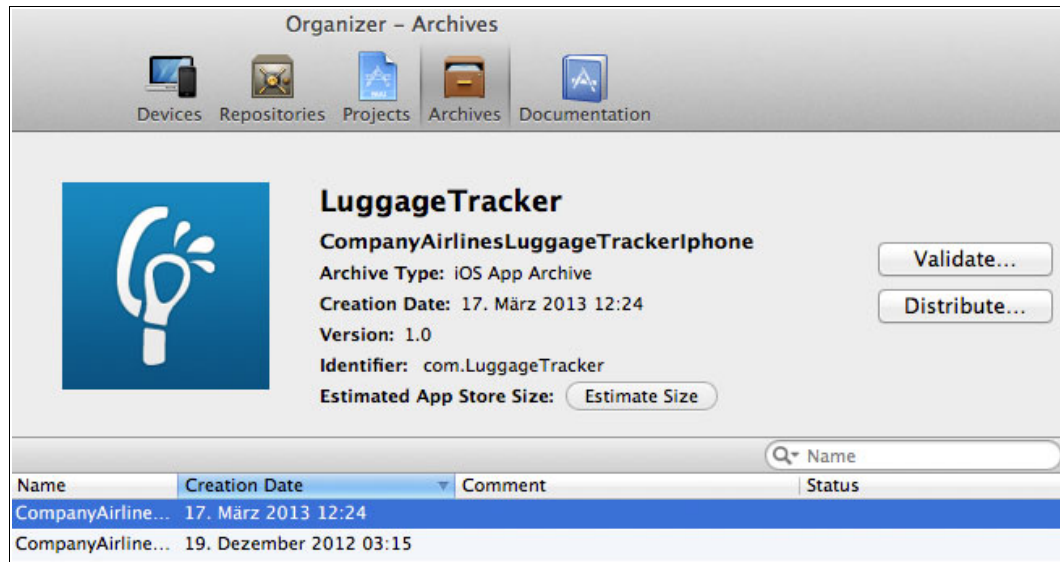


Figure 4-75 Xcode Organizer showing the LuggageTracker application

7. A wizard is displayed. On the first page of the wizard, select **Save for Enterprise or Ad-Hoc Deployment**, because LuggageTracker is an application that will be deployed and made available to company employees through Application Center rather than through the iOS App Store. Then click **Next**.
8. Select the same developer account from the Code Signing Identity list (Figure 4-73 on page 168) and then click **Next**.
9. A dialog (Figure 4-76) is opened. Specify the location to save the application file (LuggageTracker.ipa) in the **Where** field.

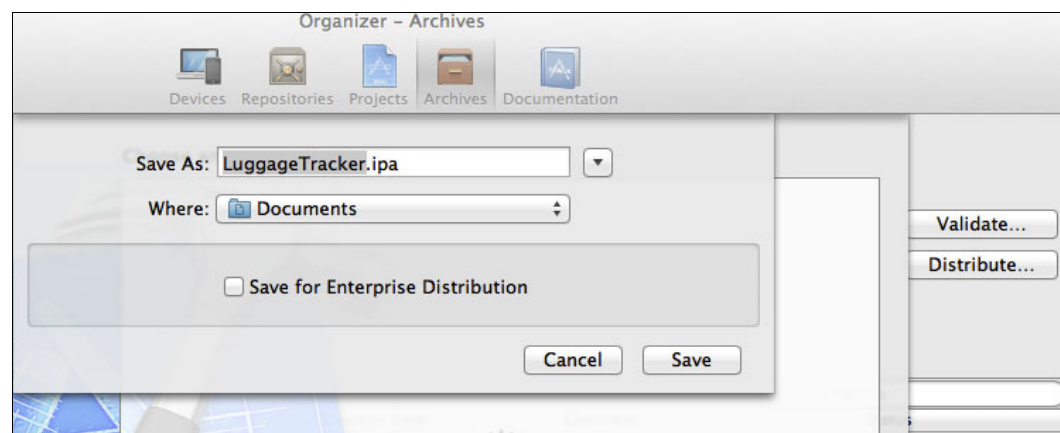


Figure 4-76 Selecting the location for the output LuggageTracker.ipa file

10. Because we specified to **Save for Enterprise or Ad-Hoc Deployment** in Step 7 on page 169, select **Save for Enterprise Distribution**. This step expands the dialog to show additional fields.

11. Enter the Application URL and then enter LuggageTracker for the Title.

12. Click **Save**.

The signed iOS application file, LuggageTracker.ipa, is created and stored in the directory that you specified in Step 9 on page 169. This ipa file is used to deploy the application to Application Center.

4.9 Deploying the application

When the new application or an application update is available, it must be deployed using IBM Worklight Server. LuggageTracker has four distinct parts to be deployed:

- ▶ Hybrid application packages for Android and iOS, deployed using Application Center
- ▶ A project-specific web application archive, deployed using the Worklight Server console
- ▶ Worklight applications, deployed using the Worklight Server console
- ▶ Worklight adapters, deployed using the Worklight Server console

In addition, if the application being deployed is dependent on new or changed business services, the system administrator must ensure that these services are available prior to deploying the application and adapters to Worklight Server.

4.9.1 Deploying the hybrid application packages

The two hybrid application packages (an .apk file for Android devices and an .ipa file for iOS devices), which were built in 4.8.6, “Building the mobile application” on page 160), must now be deployed to Application Center, Airline Company A’s internal application store. When deployed to Application Center, the CSRs can use the Application Center mobile client to download and install the applications. The system administrator receives the application packages (they typically come from either the development team or someone responsible for building the client applications) and is responsible for deploying them.

To illustrate multiple versions of an application, LuggageTracker was built twice: one for a beta version and once for a production version. These two versions are shown in several figures that follow.

Deploying the hybrid application packages has the following steps:

- ▶ Deploying the application packages
- ▶ Restricting access
- ▶ Deploying a new version

Deploying the application packages

The hybrid application packages (the .ipa and .apk files) are deployed to Application Center so that the applications can be downloaded and installed from the Application Center mobile client. In this example, the CSRs download and install the application on their mobile devices.

Use the following steps to deploy the two hybrid application packages (one for Android and one for iOS):

1. From Worklight Server, start Application Center by navigating to this address:
<http://localhost:9080/applicationcenter>
2. Log in to Application Center with the appcenteradmin user ID and password that are defined in the Worklight Server server.xml file (see “Worklight Server server.xml file” on page 323).
3. On the Applications tab, click **Add Application**.
4. Click in the **File** field to display the File Upload dialog. Navigate to the directory that contains the Android .apk file, select it, and then click **Open**. This uploads the file and displays a confirmation, as shown in Figure 4-77.

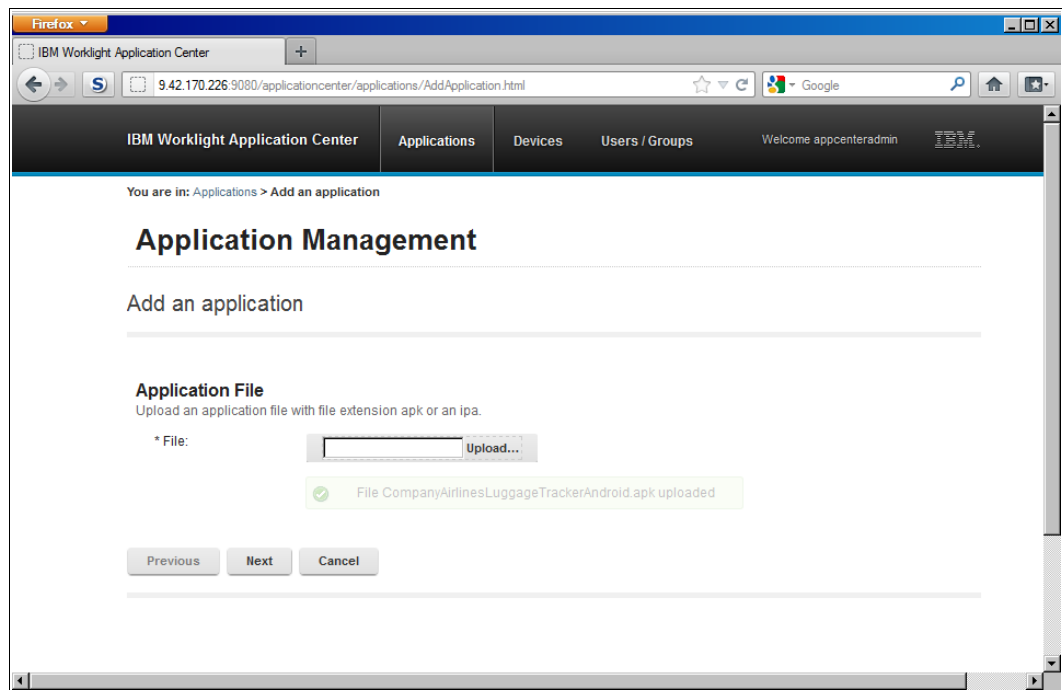


Figure 4-77 Confirmation that the application was uploaded

5. Click **Next** to display the Application Details screen. Keep the default properties that are displayed and then click **Finish**. This deploys the Android application.
6. Repeat these steps to deploy the iOS .ipa file.

The uploaded applications are now displayed in the Available Applications screen, as shown in Figure 4-78. In this example, the iOS application has not yet been deployed but both the LuggageTrackerBeta and LuggageTracker applications for Android are available.

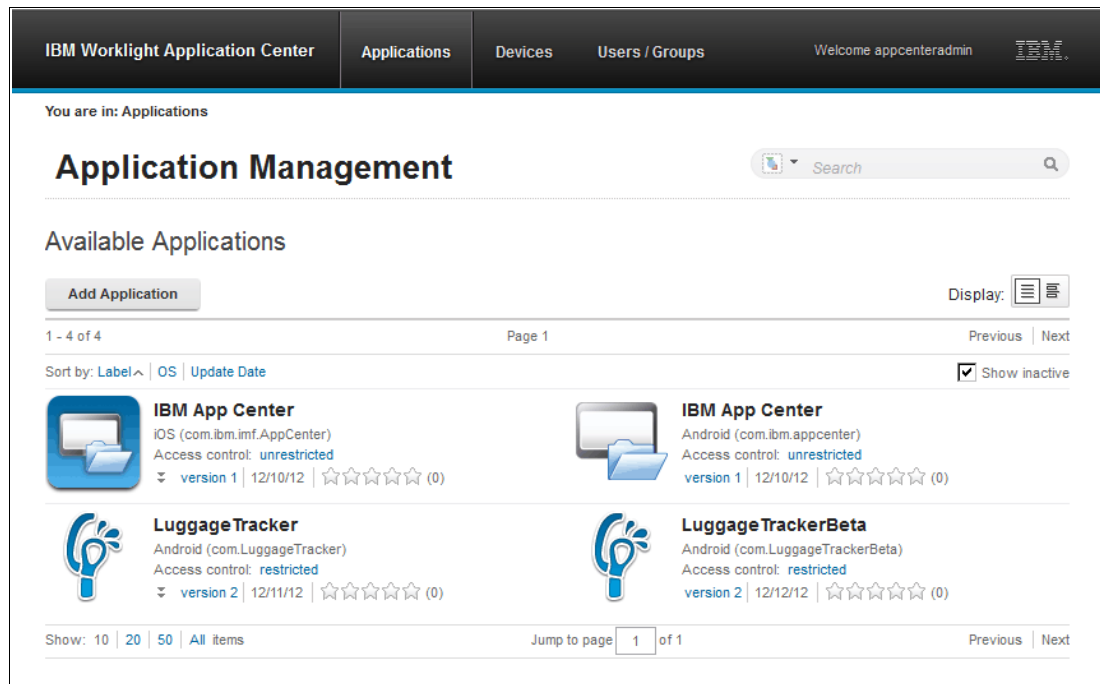


Figure 4-78 Application list showing uploaded applications

Restricting application access

By default, applications have unrestricted access, which means that anyone who logs in to Application Center can download them. In this scenario, there is a beta version of the application, which will be available only to a subset of users, and the production version, which will be available to everyone.

To restrict access to the beta version of LuggageTracker, use the following steps:

1. In the LuggageTrackerBeta section of the Available Applications screen, click the **unrestricted** link shown in Figure 4-79.

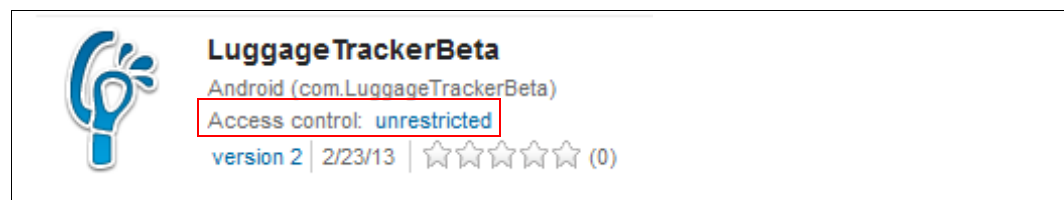


Figure 4-79 Starting point for changing the access control for an application

2. Select the **Access control enabled** check box and then add the CSR-Early Adopters and Testers groups to the list of users and groups that have access to the application. To do this, type the group name and click **Add**, as shown in Figure 4-80 on page 173. This step limits access to the application to only the members of these groups.

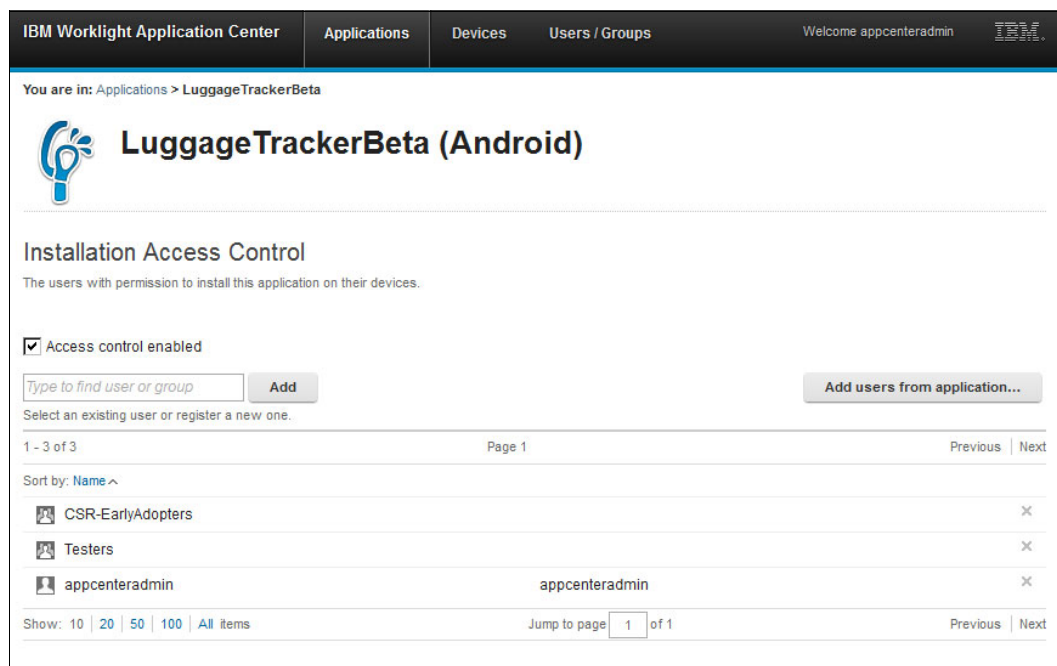


Figure 4-80 Setting the access controls for an application

The LuggageTracker and LuggageTrackerBeta applications are available to CSRs with proper access control, and they can now download and install these application using the Application Center mobile client.

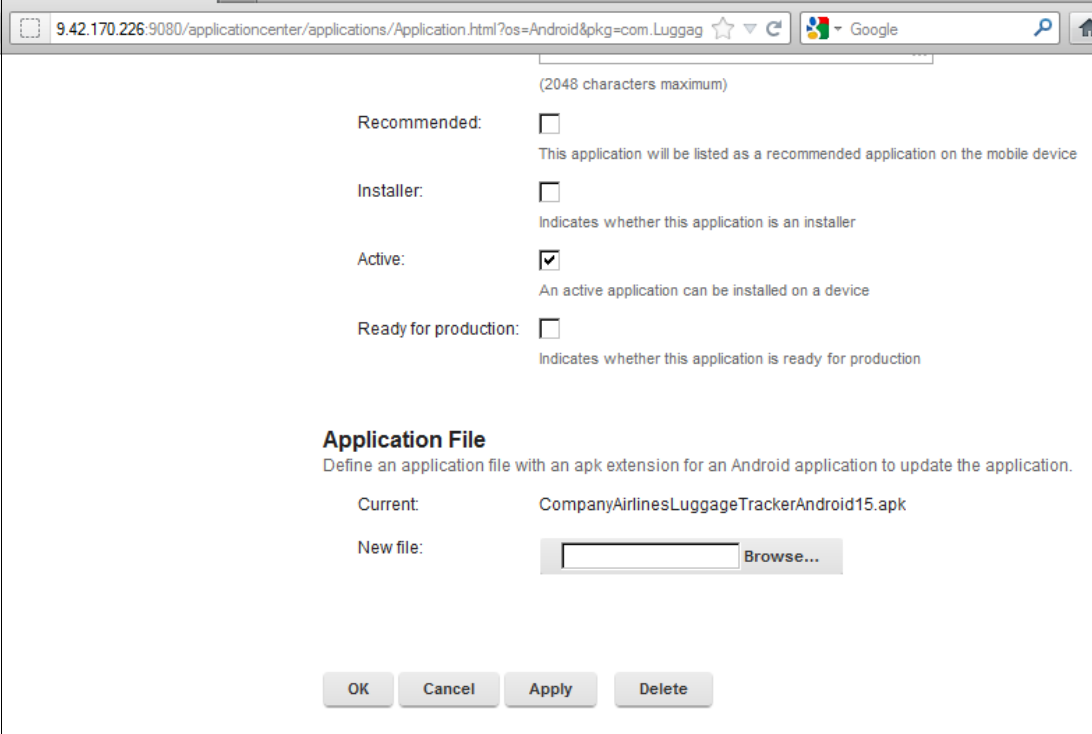
Deploying a version update

To extend the lifecycle of the application, when an updated version is available, the build and deploy processes are repeated to make the new version available in Application Center.

To deploy an updated version of the application, use the following steps:

1. From Worklight Server, start Application Center by navigating to this address:
<http://localhost:9080/applicationcenter>
2. Log in to the Application Center with the appcenteradmin user ID and password that are defined in the Worklight Server server.xml file (see “Worklight Server server.xml file” on page 323).
3. On the Applications tab, show the Application Details for the mobile application that needs to be updated.

4. On the Application details page, shown in Figure 4-81, scroll to the bottom of the page and upload the new application package.



(2048 characters maximum)

Recommended: ☐
This application will be listed as a recommended application on the mobile device

Installer: ☐
Indicates whether this application is an installer

Active: ☒
An active application can be installed on a device

Ready for production: ☐
Indicates whether this application is ready for production

Application File
Define an application file with an apk extension for an Android application to update the application.

Current: CompanyAirlinesLuggageTrackerAndroid15.apk

New file: Browse...

OK Cancel Apply Delete

Figure 4-81 Uploading a new version of the application on the Application Details page

5. Click **OK** to save your changes, which makes the updated version available.

4.9.2 Deploying the project-specific web application archive

The project-specific web application archive is deployed using the Worklight Server console. The archive contains the project configuration, including the Worklight server URL, Worklight context root, application authenticity settings, and user authentication settings that were set in the `application-descriptor.xml` file by the mobile application developer.

Deploy the web application archive, named `CompanyAirlines.war`, which was created as part of 4.8.6, “Building the mobile application” on page 160, by using the following steps:

1. The pre-production server settings, including the Worklight port number and database properties, are stored in the `worklight.properties` file in the `worklight.war` file. This `worklight.properties` file must be placed in `CompanyAirlines.war` so that these same properties are available within the Company Airlines application archive (`.war` file). Use the following steps:
 - a. Copy the `worklight.war` file from the Worklight server (located in the `<wl_install_dir>/server/wlp/usr/servers/worklightServer/apps` directory (where `<wl_install_dir>` is the base installation directory)) to a temporary location.
 - b. Extract the copied `worklight.war` file.
 - c. In the folder that is created when the `worklight.war` file is extracted, navigate to the `WEB-INF/classes/conf/` directory and open the `worklight.properties` file.

- d. Update the file's `publicWorkLightContext` property to `/luggageTracker`, which is the context root in the `application-descriptor.xml` file, as shown here:
`publicWorkLightContext=/luggageTracker`
 - e. Save and close the `worklight.properties` file.
 - f. Extract the `CompanyAirlines.war` file.
 - g. Replace the `worklight.properties` file that is currently in the `WEB-INF/classes/conf/` directory of the folder that was created when you extracted the `CompanyAirlines.war` file with the `worklight.properties` file that you modified and saved in the previous step, which contains the updated `publicWorkLightContext` property.
 - h. Re-compress the folder (that was created when you extracted the `CompanyAirlines.war` file) to a temporary name, such as `CompanyAirlines-updated.war`.
 - i. Rename the updated `CompanyAirlines.war` file from step 1h to `luggageTracker.war`. This causes the file name to match the context root (which is now specified in both the `application-descriptor.xml` file and the updated `worklight.properties` file).
2. Copy the new `luggageTracker.war` file to the IBM Worklight server in the `<wl_install_dir>/server/wlp/usr/servers/worklightServer/apps` directory (where `<wl_install_dir>` is the directory where Worklight Server is installed).
 3. Declare the web application in the application server (the Liberty Profile server in this scenario) as follows:
 - a. Edit the `server.xml` file for Worklight Server, which is located in following directory (where `<wl_install_dir>` is the directory where the Worklight Server is installed):
`<wl_install_dir>/server/wlp/usr/servers/worklightServer/`
 - b. Add the XML shown in Example 4-36 to the `server.xml` file, after the section that declares the web application named `worklight` (begins `<application id="worklight"` and ends `</application>`).

Example 4-36 XML to add the LuggageTracker application to the server.xml file

```

<application id="luggageTracker" name="luggageTracker"
  location="luggageTracker.war" type="war">
  <classloader delegation="parentLast">
    <commonLibrary>
      <fileset dir="${shared.resource.dir}/lib"
includes="worklight-jee-library.jar"/>
    </commonLibrary>
  </classloader>
</application>

```

- c. Save the and close the `server.xml` file.
4. Restart the Worklight Server to activate these changes.
5. To verify that the deployment was successful, use a browser to access the Worklight Server console for LuggageTracker at the following address:
`http://localhost:9080/luggageTracker/console`

4.9.3 Deploying the Worklight application

The Worklight application (provided to the system administrator in the form of one or more .wlapp files) must be deployed on the Worklight server to make LuggageTracker fully functional. The Worklight application contains the web artifacts associated with the hybrid application and enables you to update the artifacts without requiring the entire mobile application to be reinstalled.

Two steps are involved in deploying the Worklight application, which are detailed in the following subsections:

- ▶ Deploying the application packages
- ▶ Deploying a new version

Deploying the application packages

In this scenario, the system administrator has three .wlapp files for LuggageTracker (similar files were also received for LuggageTrackerBeta) to deploy:

- ▶ LuggageTracker-common.wlapp, containing the web code common to both Android and iOS platforms
- ▶ LuggageTracker-android-1.0.wlapp, containing the specific web code for Android
- ▶ LuggageTracker-iphone-1.0.wlapp, containing the specific web code for iOS

Use the following steps to deploy the Worklight application files for LuggageTracker and LuggageTrackerBeta:

1. From Worklight Server, start the LuggageTracker console, (shown in Figure 4-82) using the following address:

`http://localhost:9080/luggagetracker/console`

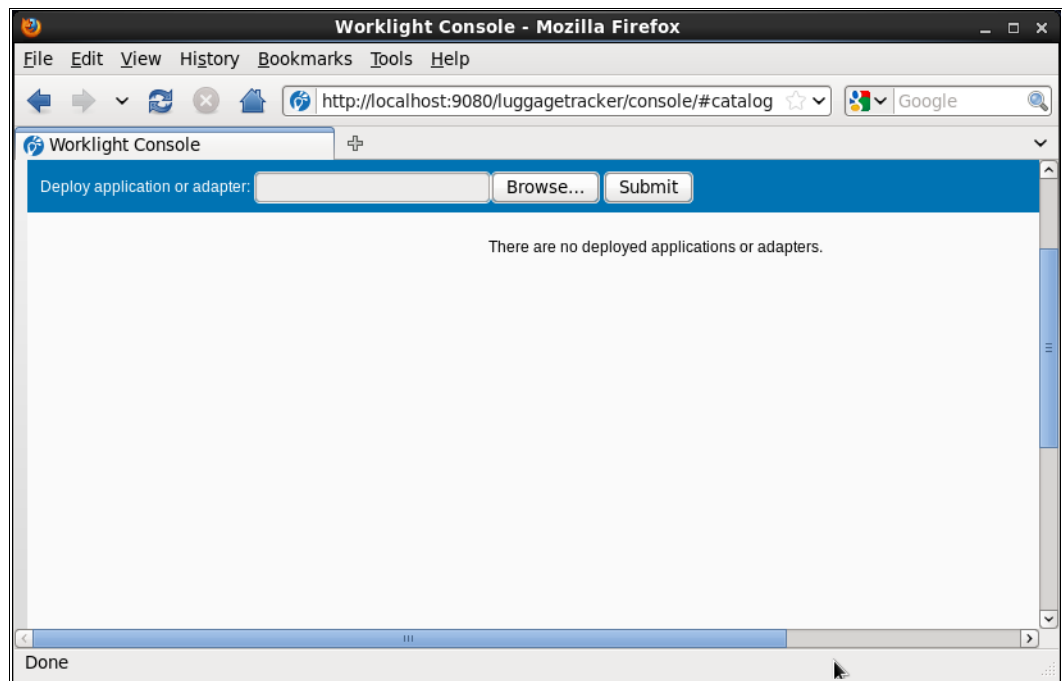


Figure 4-82 Worklight console for LuggageTracker

2. Click **Browse**, then navigate to the LuggageTracker-common.wlapp file, select it, and click **Open**. This steps puts the file name in the text field, as shown in Figure 4-83.

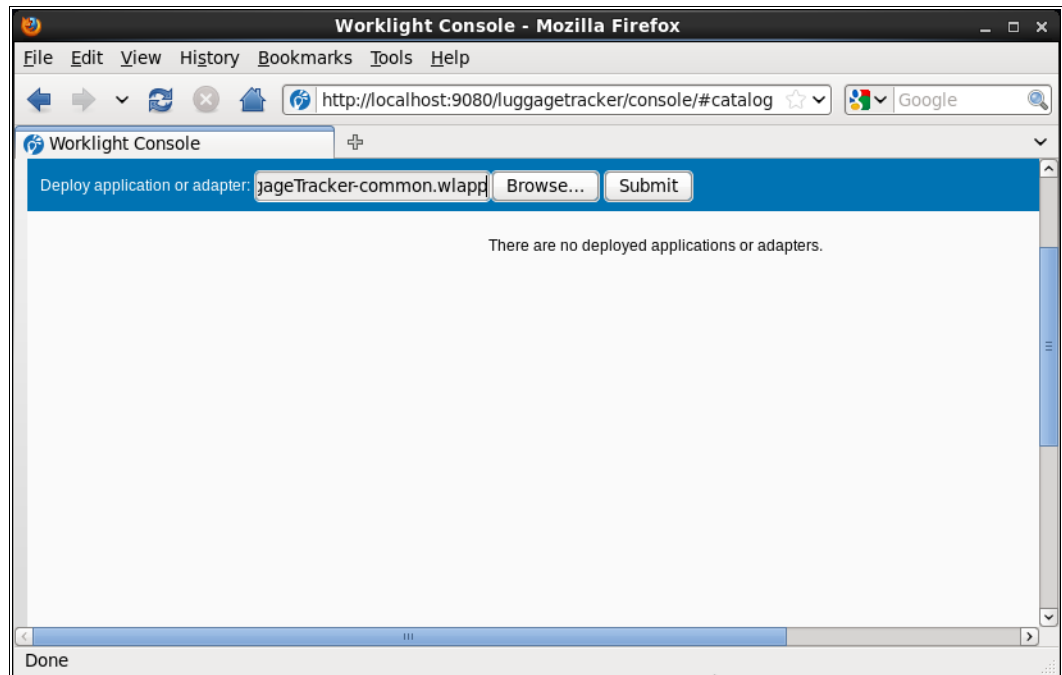


Figure 4-83 Submitting to upload and deploy the application

3. Click **Submit** to upload and deploy the Worklight application.
4. Repeat steps 2 and 3 for two remaining .wlapp files for LuggageTracker.
5. Repeat steps 1 on page 178 through 3 for the .wlapp files for LuggageTrackerBeta.

The installed applications are displayed in the catalog, shown in Figure 4-84.

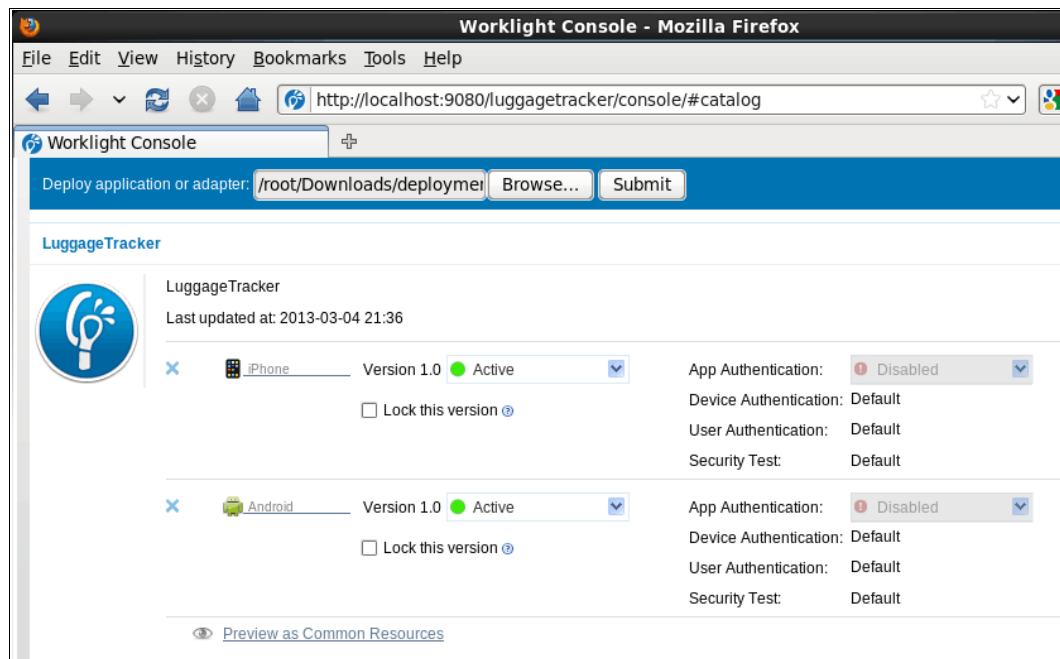


Figure 4-84 Deployed applications shown in the LuggageTracker console

Deploying a version update

To extend the lifecycle of the application, when an updated version of the application is available, the building and deployment processes are repeated by the system administrator to deploy an updated version of the Worklight application to Application Center. The individual steps will be not repeated again here.

4.9.4 Deploying the Worklight adapters

LuggageTracker has two adapters, both of which must be deployed for the application to operate:

- ▶ BusinessServicesAdapter
- ▶ LuggageTrackerAdapter

These adapters are created in 4.7, “Creating the adapters” on page 88 and built in “Building the adapters” on page 162. To deploy the adapters, use the following steps:

1. From the Worklight server, start the LuggageTracker console using the following address:
`http://localhost:9080/luggagetracker/console`
2. On the Catalog tab, click **Browse**.
3. Navigate to the `BusinessServicesAdapter.adapter` file, select it, and then click **Open**. The file name is displayed in the text field, as shown in Figure 4-85.

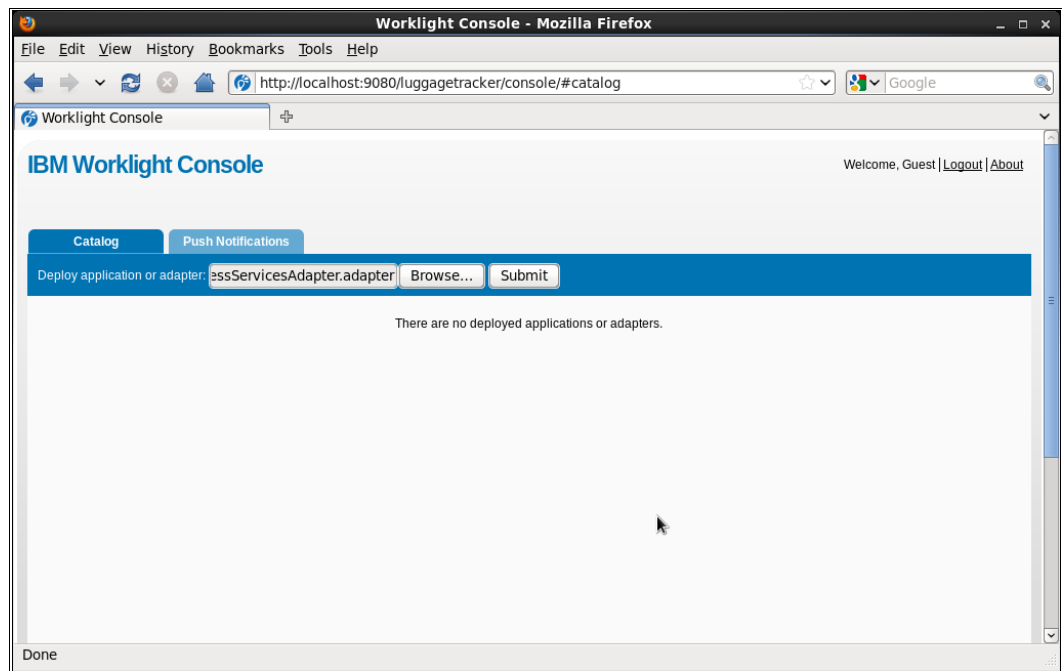


Figure 4-85 Selecting the adapter file

4. Click **Submit**. The adapter is uploaded to Worklight Server and deployed, and a confirmation message is displayed on the Worklight console, as shown in Figure 4-86.

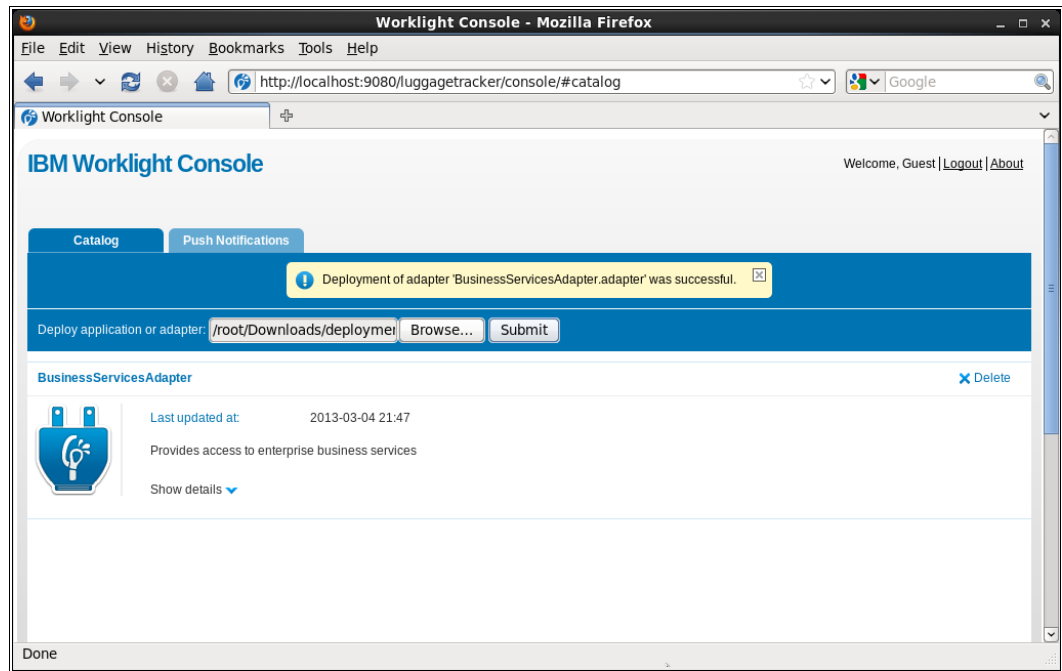


Figure 4-86 Message confirming successful deployment of adapter

5. Repeat steps 2 on page 178 through 4 to deploy the `LuggageTrackerAdapter.adapter` file.
- When a new version of the adapter is available, update it by following these same steps.

4.10 Testing the end-to-end scenario

Finally, the application is downloaded and tested on actual mobile devices, including the act of downloading the application from Application Center. This allows the testers to duplicate the same user experience of the CSRs that will use the application in the field.

The mobile testers use the following steps:

1. Install the Application Center mobile client.
2. Install the LuggageTracker application.
3. Start the application and test each anticipated usage scenario.

4.10.1 Installing the Application Center mobile client

Each tester starts by downloading and installing the Application Center mobile client using the Worklight server IP address (pre-production environment) and individual credentials the testers are assigned.

Complete the following steps:

1. Use the mobile device browser to access the following URL, where `<wl_server_ipaddress>` is the IP address of Worklight Server:
`http://<wl_server_ipaddress>:9080/applicationcenter/installers.html`
2. Authenticate with Application Center (using the credentials created in 4.5.1, “Managing access to the Application Center” on page 78). In Figure 4-87, the user is submitting the credentials for CSR1.

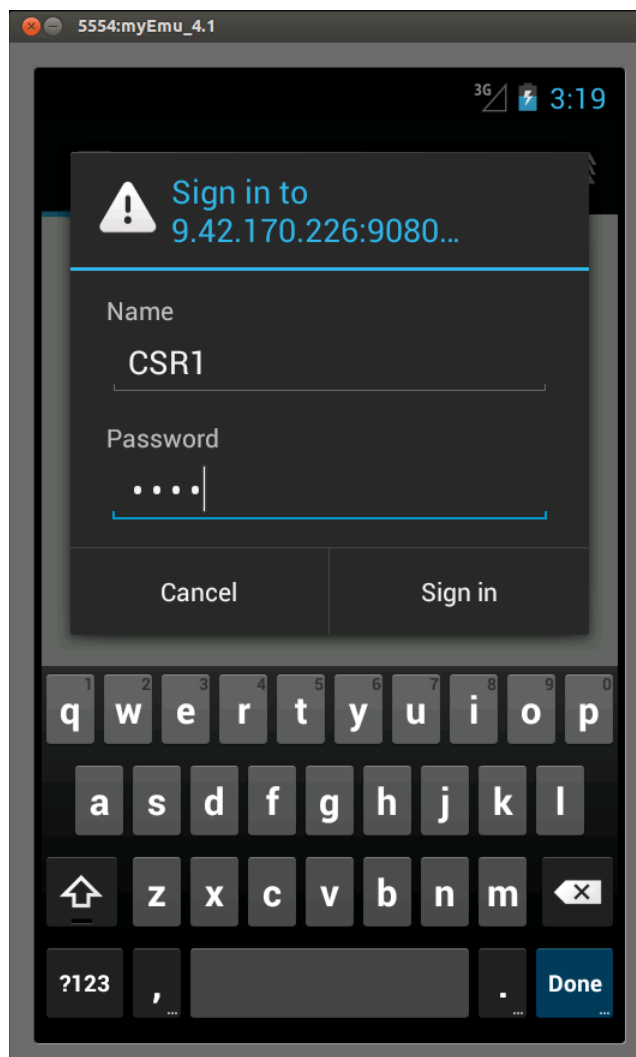


Figure 4-87 Authenticating with Application Center from the mobile device

3. When access is granted, a list of available applications is presented, as shown in Figure 4-88.

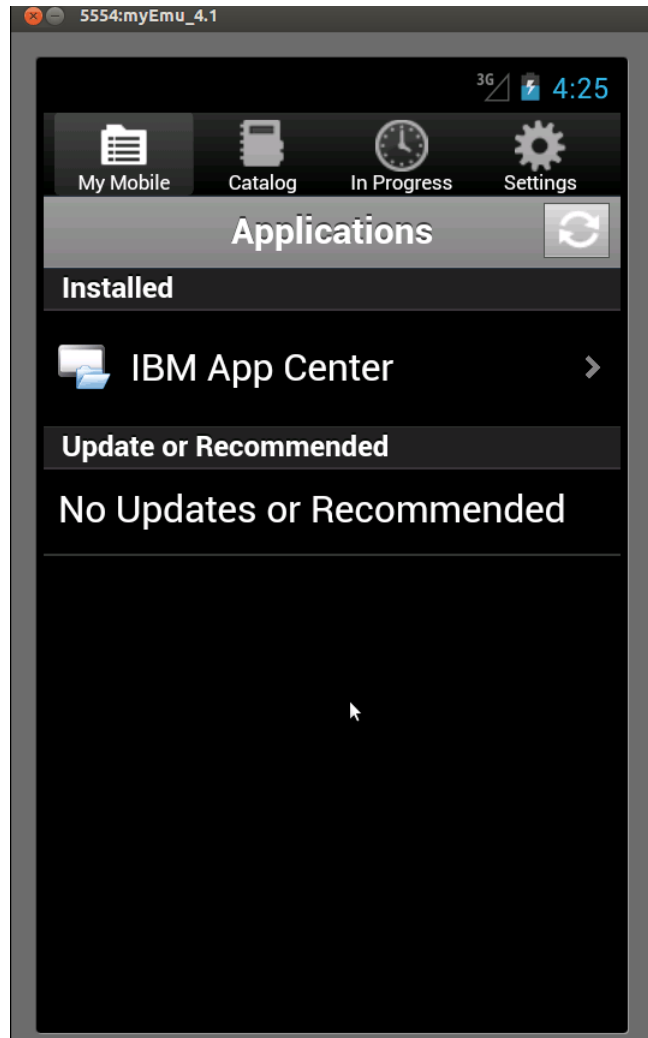


Figure 4-88 List of available applications that can be installed

4. Select the IBM Worklight Application Center mobile client, named IBM App Center, to view details about the application, as shown in Figure 4-89.

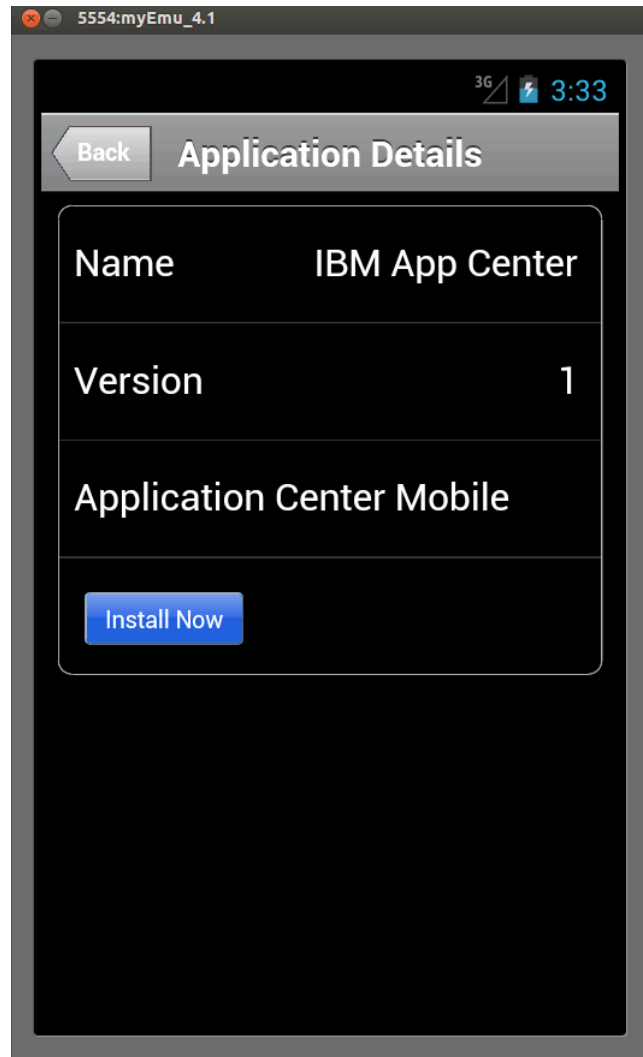


Figure 4-89 Application details for the mobile client

5. Click **Install Now** to begin the installation process. The mobile client package is downloaded and a confirmation is displayed (Figure 4-90).

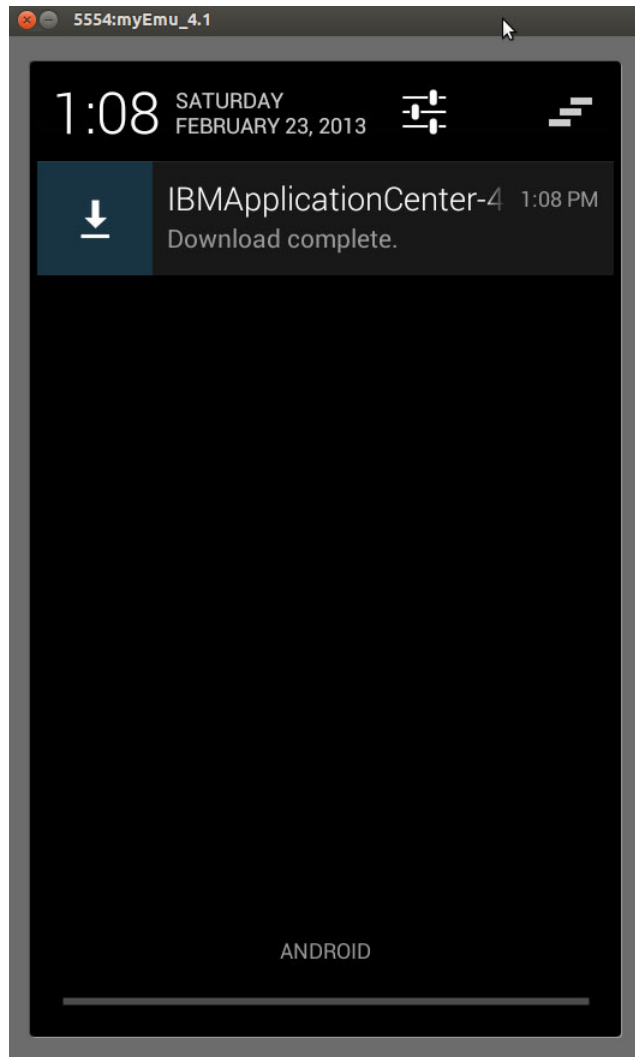


Figure 4-90 Confirmation of completed application download

6. Select the downloaded file from the notification area to install it. The installation screen opens and presents the mobile device access rights that will be granted to IBM App Center, as shown in Figure 4-91.

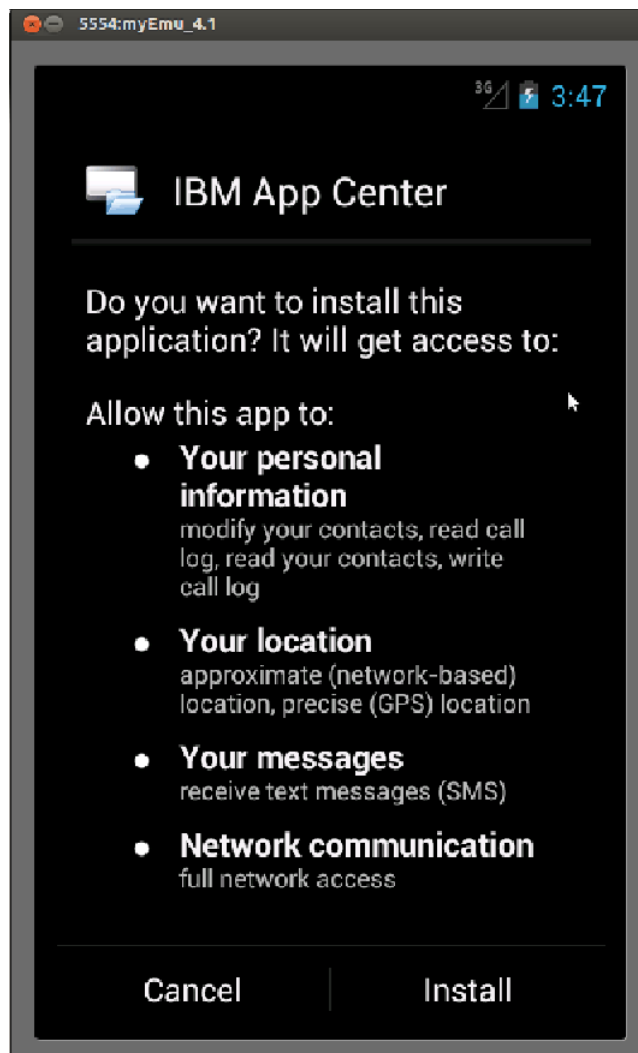


Figure 4-91 Display of application access rights and Install and Cancel buttons

7. Click **Install** to install the mobile client.
8. The Application Center mobile client is now installed.

4.10.2 Installing the mobile application

LuggageTracker is installed from Application Center using the mobile client. To install the mobile application, use the following steps:

1. Start the Application Center mobile client.
2. On the Settings screen, shown in Figure 4-92, enter valid CSR credentials in the Username and Password fields. Because this is the first time you are using the mobile client and the Application Center URL is not yet saved in the mobile client, provide the URL to the Application Center (http://<wl_server_ipaddress>:9080/applicationcenter, where *<wl_server_ipaddress>* is the IP address of Worklight Server) in the Server field and then click **Connect**.

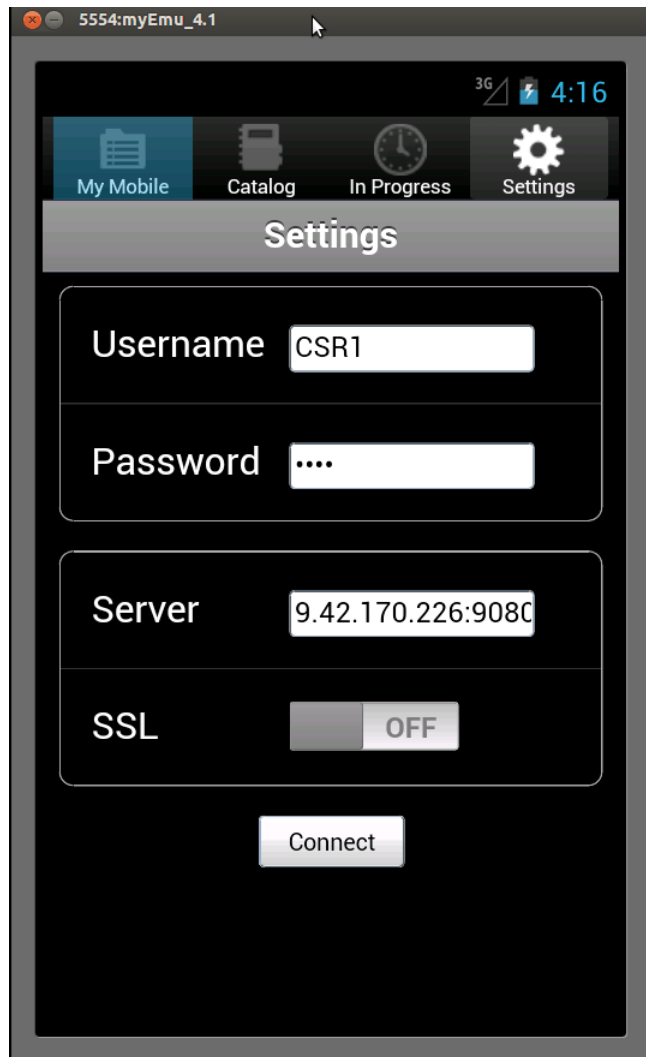


Figure 4-92 Specifying the username, password, and server IP address

3. After the mobile client connects to Application Center and the credentials are validated successfully, touch the Catalog icon to display the available applications that the authenticated user has access rights to, as shown in Figure 4-93.

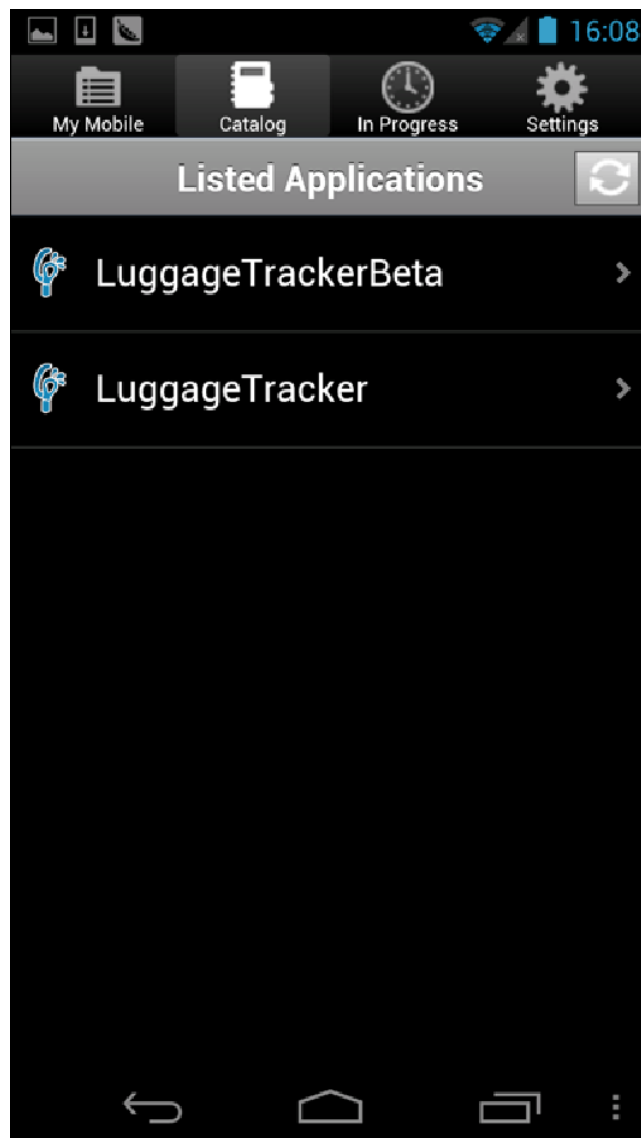


Figure 4-93 Application catalog showing available applications

4. Select LuggageTracker by touching the application name on the screen. You are asked if you want to install the application (Figure 4-94).

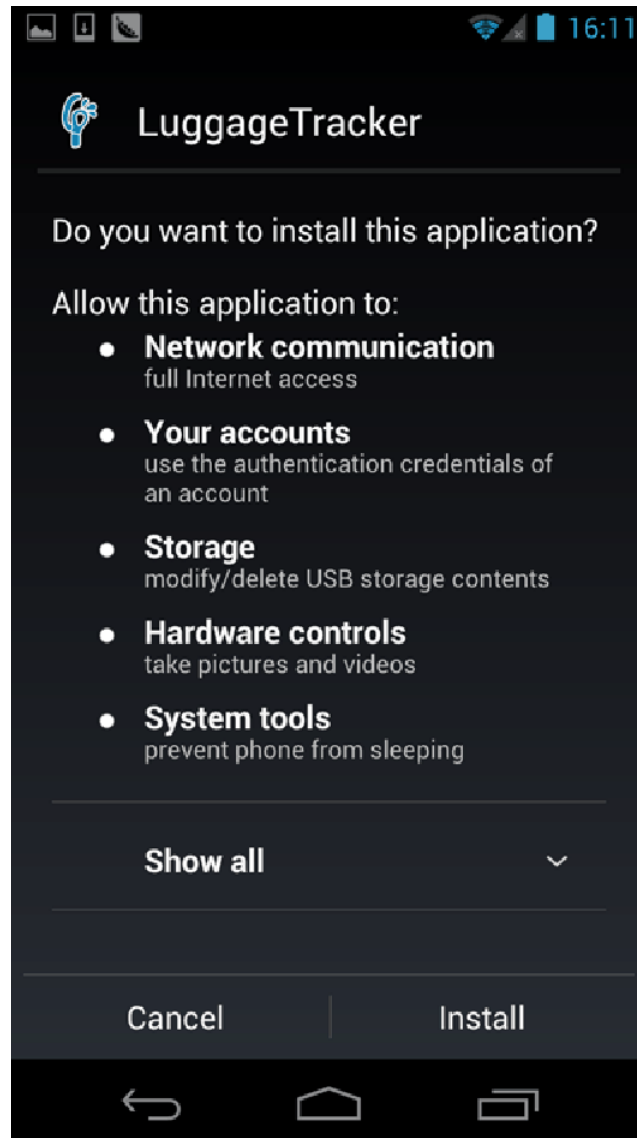


Figure 4-94 Confirmation to install application, including list of permissions to the mobile device

5. Click **Install** to install the application on the mobile device.

4.10.3 Starting the application and performing the tests

After the application installation is complete, the mobile tester starts LuggageTracker on the mobile device and performs the required tests to confirm the functionality of the application from a user (CSR) perspective. Use the following steps:

1. Start LuggageTracker to display the login screen, as shown in Figure 4-95.

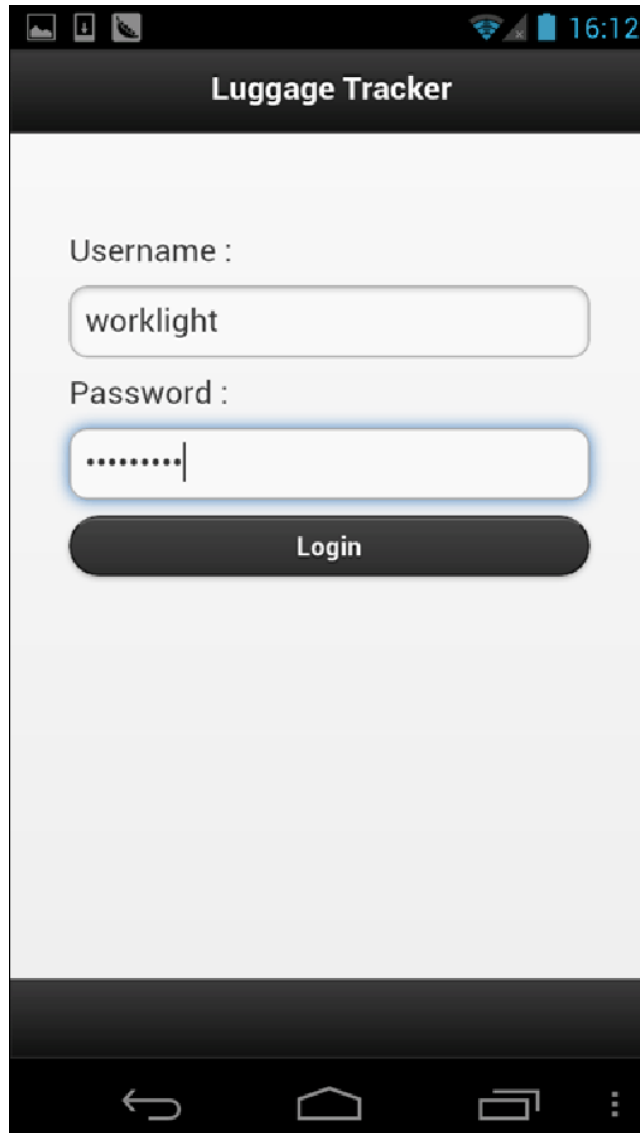


Figure 4-95 LuggageTracker login screen on actual mobile device

2. Enter valid credentials (provided by the system administrator) and click **Login**.
3. The Luggage ID input screen is displayed.

All required scenarios can now be done to test the end-to-end functionality using a live mobile device. The exact scenarios and steps to run each scenario are not covered here.

4.11 Using feedback mechanisms

One of the expected key performance indicators (KPIs) for Airline Company A's mobile strategy is the feedback from the CSRs who use the new application, and the ratings they give it. This information helps the development team improve the application in later releases.

The CSRs can rate and provide feedback on the application using the Application Center mobile client, as shown in Figure 4-96. These feedback functions are built into the mobile client and do not need to be developed from scratch by the development team.

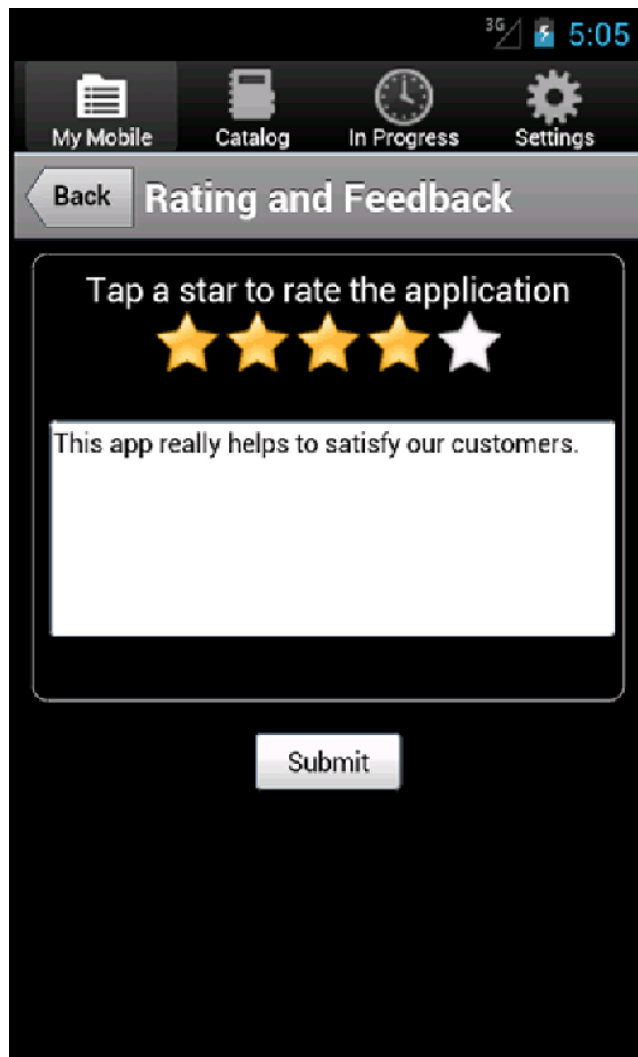


Figure 4-96 Feedback mechanism within the Application Center mobile client

The feedback and rating data are stored and viewed based on the application and device platform. So for LuggageTracker, which supports both Android and iOS, the feedback for the Android-based application is separate from the feedback for the iOS-based application.

The feedback statements and rating data are sent to Worklight Server and can be viewed in Application Center by users with the appcenteradmin role (normally system administrators).

To view the feedback and ratings that have been provided by CSRs for LuggageTracker, use the following steps:

1. From Worklight Server, start Application Center using the following address:
`http://localhost:9080/applicationcenter`
2. Sign in using a valid user name and password that has the `appcenteradmin` role (for example, the user `appcenteradmin` and password `admin`).
3. Click the **Applications** tab to view the installed applications.
4. Select the **LuggageTracker (Android)** application to see the feedback and rating for the Android-based LuggageTracker application. The application name that is used here is slightly different from usual because of the need to segment the feedback received for the Android version from the feedback received for any other versions.
5. Click **Feedback**. The feedback received from CSRs running LuggageTracker on an Android device is displayed, along with the overall rating for the application provided by all users on all platforms, as shown in Figure 4-97.

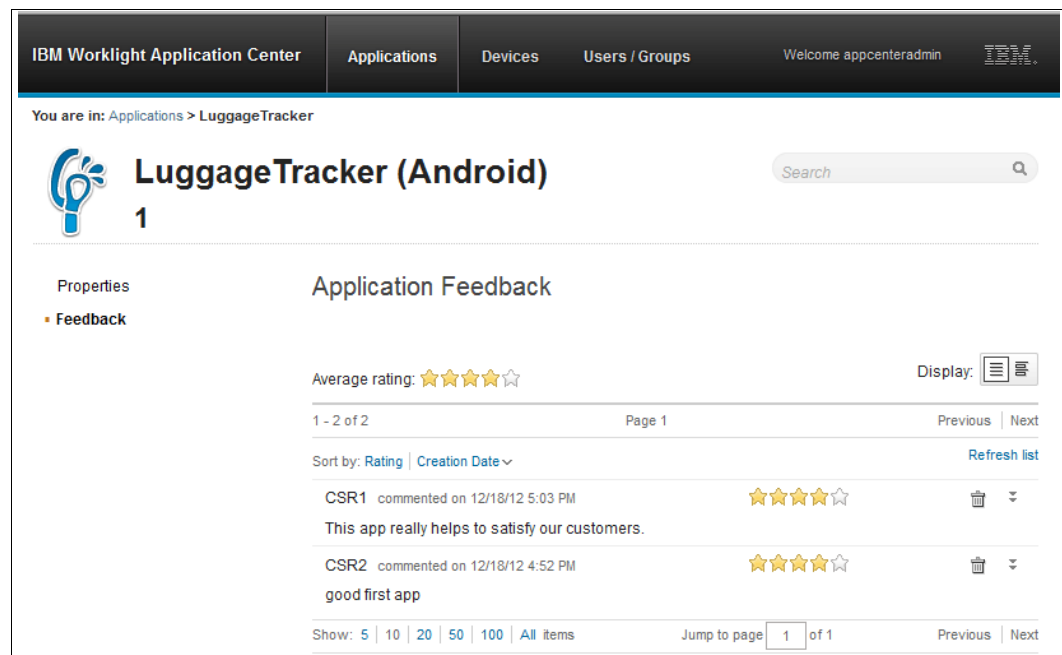


Figure 4-97 Feedback and rating for the LuggageTracker (Android) application

The displayed feedback refers to a specific version of the application (for example, LuggageTracker on Android versus LuggageTracker on iOS), whereas the rating is an average of all the feedback received for all versions of an application over time. Users can rate the application from one (low) to five (high) stars. If a user enters a comment, even a negative one, but does not provide a rating, nothing is registered for the rating and the average is not affected. Comments can be deleted from the database that they are stored in using the trash can icon.

The feedback and rating information is used by the mobile application development team to improve the application with bug fixes and new features, as part of the continuous development lifecycle. Airline Company A is using the SCRUM development methodology (as mentioned in 2.1.1, “Building mobile applications” on page 22), with one *sprint* equating to one application version release. This feedback is part of the *sprint review* and, through the *sprint retrospective*, is used as input to the next sprint where an updated version of the application is created and deployed.

4.12 Analyzing application and adapter usage

Worklight Server provides extensible reporting and analytics capabilities. The reporting component structure of IBM Worklight is shown in Figure 4-98. In addition to the raw data, Worklight Server periodically creates usage reports by running aggregations on the raw data. Both the raw and aggregated data can be accessed using a company's own reporting tools or integrate it with third-party business intelligence software. In addition, IBM Worklight provides predefined reports for use with the Business Intelligence and Reporting Tools (BIRT) project from Eclipse.

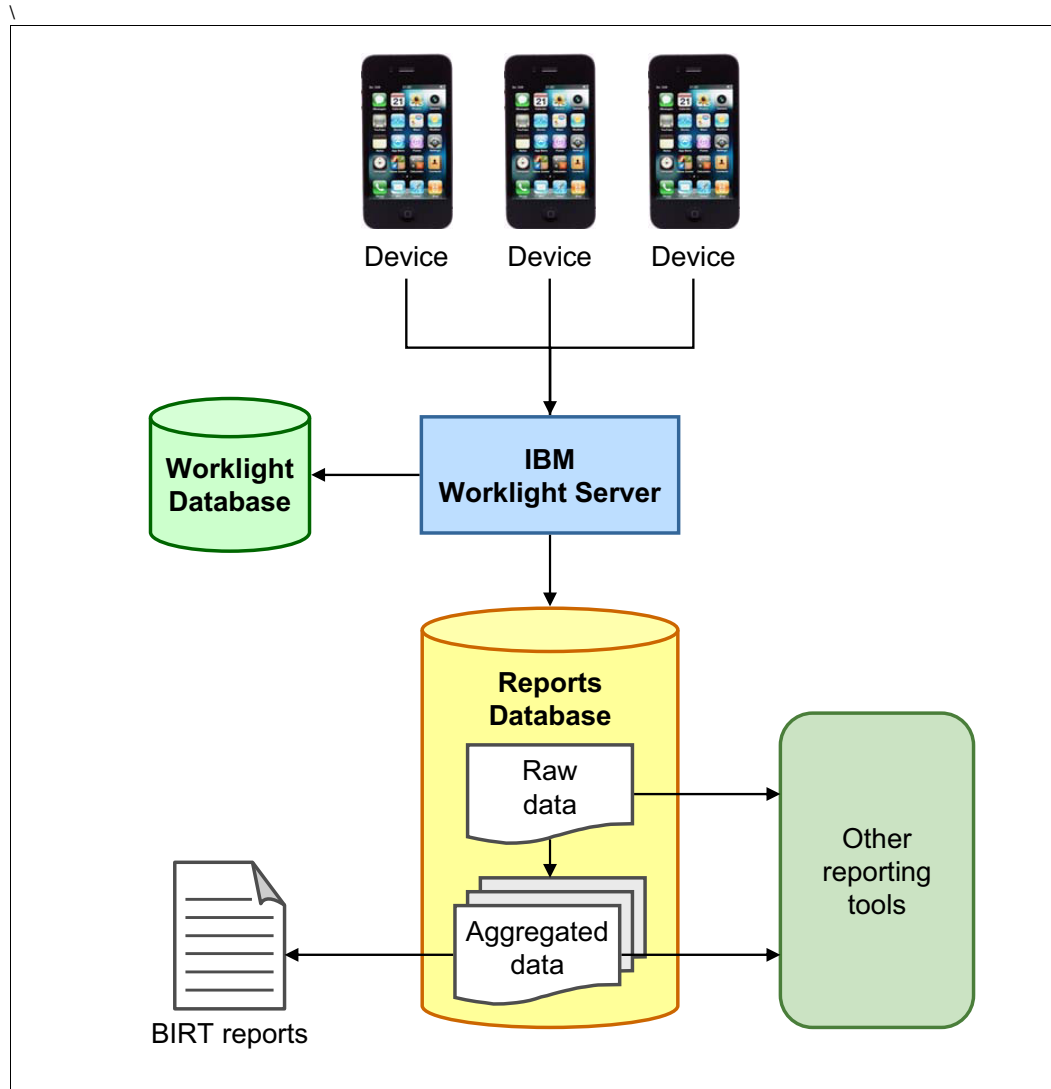


Figure 4-98 IBM Worklight reporting component structure

The mobile application (running on the mobile devices) uses the IBM Worklight Reporting API to create activity logs when certain application functions are called. The mobile application then sends these activity logs at run time to Worklight Server, where they are stored in a separate reporting database. The Worklight adapters also generate reporting events that are stored in the same reporting database each time the adapters are called.

Worklight Server stores this raw data in a table named `app_activity_report` in the reporting database. The raw data comprises the following information:

- ▶ Device environment (such as Android or iOS)
- ▶ User identifier
- ▶ Worklight adapter name
- ▶ Name of called procedure
- ▶ HTTP user agent of device
- ▶ Session ID
- ▶ IP address of client
- ▶ Unique device ID
- ▶ Device mode
- ▶ Device operating system, including version number

Airline Company A decided to use the predefined BIRT reports during the initial phase of the mobile roadmap.

4.12.1 Configuring the reporting feature

By default, the Worklight Server reporting feature is disabled and any events that are sent by an application are not stored. To enable these reporting functions, use the following steps:

1. Stop the Worklight server.
2. Locate the Worklight web application archive (`worklight.war`) located in the `<wl_install_dir>/server/wlp/usr/servers/worklightServer/apps` directory (where `<wl_install_dir>` is the directory where the IBM Worklight Server is installed).
3. Extract the `worklight.war` file and edit the `worklight.properties` file (located in the `/WEB-INF/classes/conf` folder of the unzipped WAR file).
4. To activate the reporting feature, un-comment the `reports.exportRawData` property and set its value to `true`, so that the line reads as follows:

```
reports.exportRawData=true
```

5. Save the `worklight.properties` file.
6. Re-compress the `worklight.war` file.
7. Start the Worklight server.

On a defined interval, Worklight Server creates usage reports by running aggregations on the raw data stored in the reporting database. These reporting aggregations run every 24 hours by default, but this processing interval can be modified by changing the following property in the `worklight.properties` file:

```
# Default interval value for analytics processing task in seconds
wl.db.factProcessingInterval=600
```

In the example, the processing interval is set to 10 minutes (600 seconds). To change this value, follow the same steps used to enable reporting:

1. Stop the Worklight server.
2. Extract the `worklight.war` file.
3. Edit the `worklight.properties` file to change the processing interval value.
4. Re-compress the `.war` file.
5. Restart the Worklight Server.

When the reporting aggregation runs, the data entries are retrieved from the `app_activity_report` table. Then, they are processed and the aggregated information is added to the `fact_activities` table. The sample data shown in the `fact_activities` table in

Figure 4-99 shows how the data is organized in the reporting database. It displays a total activity count (number of logged actions) per application and also application version, device, and environment.

ACTIVITY_DATE	APP_NAME	APP_VERSION	DEVICE_ID	ENVIRONMENT	TOTAL_ACTIVITY
2012-12-18	LuggageTracker	1.0	68417e09-c51a-4d6c-bcb1-62b9422775a2	android	3
2012-12-18	LuggageTracker	1.0	4c2e8bb6-f2d6-49b3-8e18-9c12593fd062	android	5
2012-12-18	LuggageTracker	1.0	4f4845cb-5969-4486-bd63-42d1da79e5c6	android	24
2012-12-18	LuggageTracker	1.0	4f4845cb-5969-4486-bd63-42d1da79e5c6	android	34
2012-12-18	LuggageTracker	1.0	540dd67a-bea6-4de8-ba1b-c5f2469a108a	android	5
2012-12-18	LuggageTracker	1.0	491d9d32-081c-4627-ae7d-640bf294060c	android	14

Figure 4-99 Aggregated usage reports data

4.12.2 Setting up the reporting environment

IBM Worklight Server includes a set of predefined reports to easily glean meaningful information from the data gathered by the reporting component. These reports can be used by Business Intelligence Reporting Tools (BIRT), an Eclipse-based open source reporting system.

Installing and configuring BIRT

To install and configure BIRT, use the following steps:

1. Download BIRT from the Eclipse site:

<http://download.eclipse.org/birt/downloads>

For the purpose of this book, the All-in-One package was used. If you download and install BIRT plug-ins into an existing Eclipse environment, see the BIRT documentation for usage information.

Latest version: On the download page, select the latest version, which at the time that this book was written was **BIRT Report Designer Release Build: 4.2.2**.

2. Extract the downloaded file.
3. Start BIRT Report Designer by invoking the Eclipse executable file for your platform (for example, `eclipse.exe` on Windows), which is located in the `eclipse` folder of the folder that was created when you extracted the downloaded file. The Welcome tab for the Eclipse IDE for Java and Report Developers is displayed.
4. Close the welcome tab.
5. Open the Report Design perspective by selecting **Window** → **Open Perspective** → **Other** and then choosing **Report Design** from the dialog.

Creating a report project

As with other Eclipse-based tools, BIRT manages reports using a project. To create a project using the New Project wizard, use the following steps:

1. Select **File** → **New** → **Project** to start the wizard.
2. Select **Report Project** (located under the Business Intelligence and Reporting Tools folder), as shown in Figure 4-100, and then click **Next**.

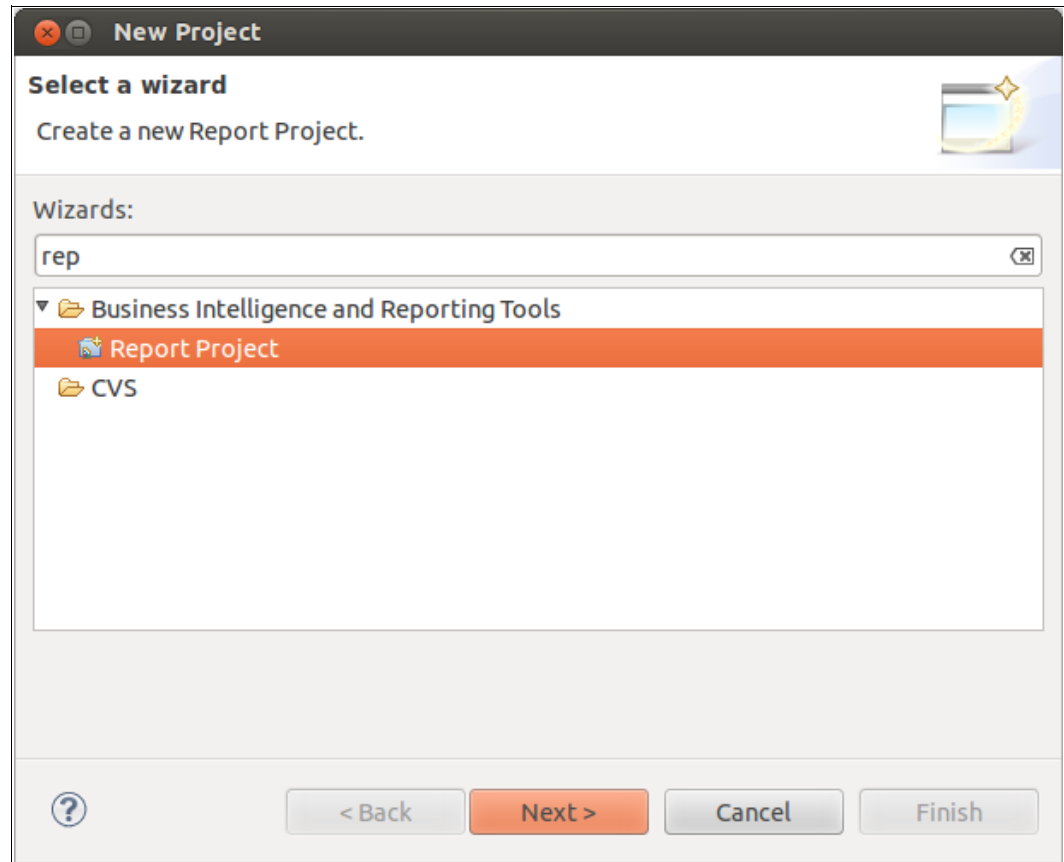


Figure 4-100 Selecting the Report Project wizard to create a Report project

3. Enter CompanyAirlineA-Reporting for the Project Name, as shown in Figure 4-101. Keep the default location and then click **Finish**.

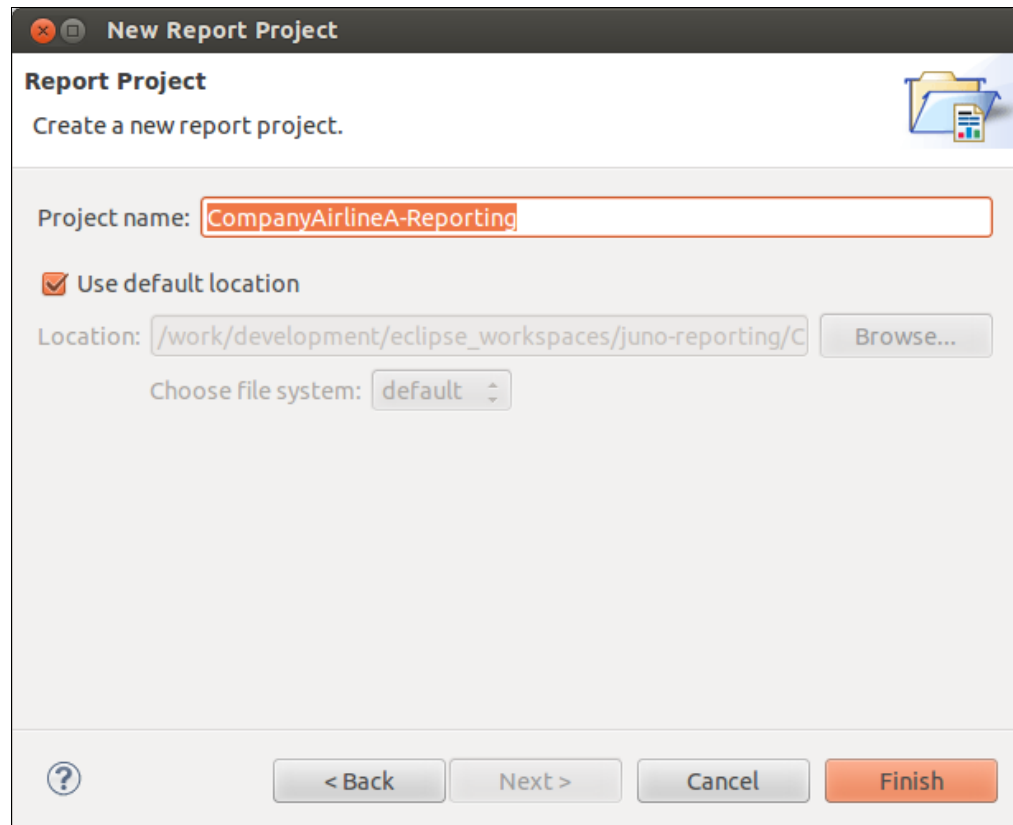


Figure 4-101 Defining the reporting project name

The project is now displayed in the Navigator view.

Importing the report templates

As previously mentioned, the IBM Worklight Server installation includes a set of predefined reports (in the form of report templates) that can be used with BIRT to view reporting data. To use these reports, you must copy them from the IBM Worklight Server installation directory to a location that is accessible from BIRT. Import the reporting templates by following these steps:

1. Copy the reporting templates to a temporary directory on your local machine. These were installed as part of the IBM Worklight Server and are located in the following directory (where `<wl_install_dir>` is the directory where IBM Worklight Server is installed):
`<wl_install_dir>/WorklightServer/report-templates`
2. Select **File** → **Import** to start the Import wizard.
3. Expand the General folder and then select **File System** for the source of the import.
4. Enter the temporary directory where you placed the copied report templates.
5. Select the **report-templates** check box in the left pane, as shown in Figure 4-102 on page 196, and then click **Finish**.

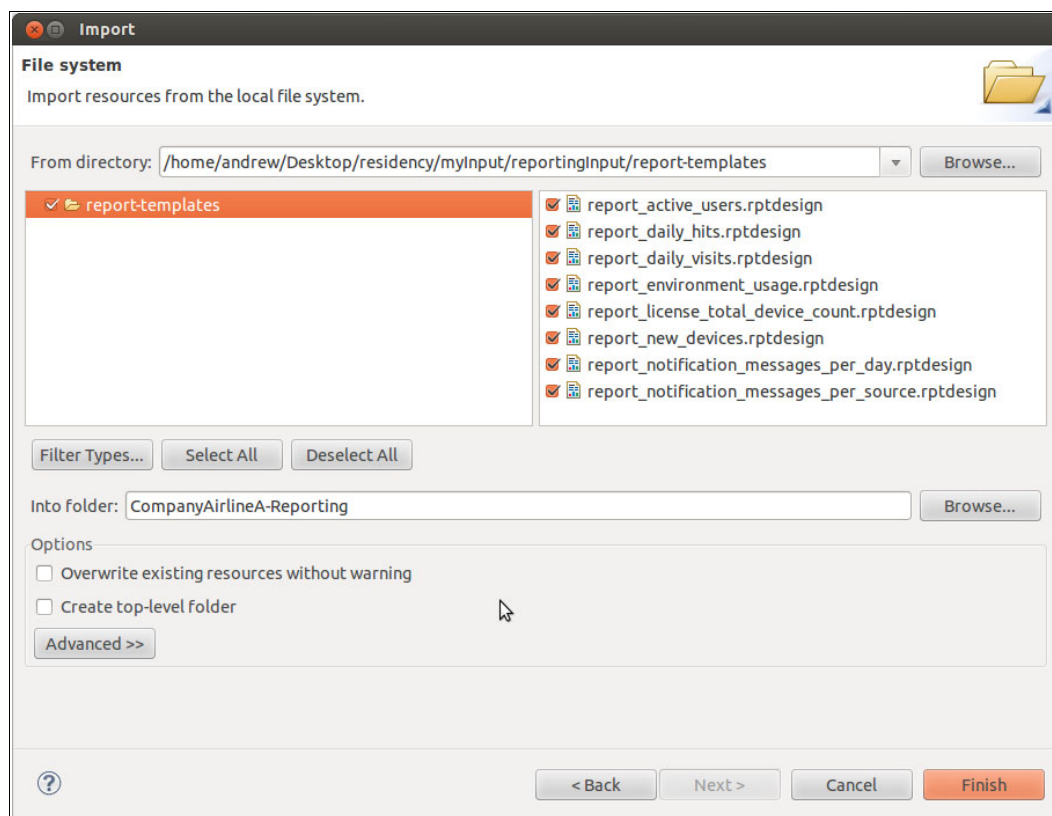


Figure 4-102 Importing the reporting templates provided by IBM Worklight Server

The Worklight Server reporting templates are now imported and can be used by BIRT.

Configuring the reports

The report templates must be modified to configure them to work in your individual BIRT environment. The reporting templates are configured to access the Worklight reporting database directly. However, because the Worklight server in this scenario uses Derby (which allows only one concurrent connection to the database), a copy of the reporting database must be created for use with the reporting tool.

Complete the following steps

1. Stop the Worklight server.
2. Copy the WLREPORT directory for Derby (which contains the reporting database) to the machine where you installed BIRT. The location of the WLREPORT database depends on the operating system of your Worklight Server installation. The server for this book was installed on Linux at `/var/ibm/Worklight/derby/WLREPORT`.
3. Start the Worklight server.

Now, update all reporting templates to use this local copy of the WLREPORT database:

1. Expand the CompanyAirlineA-Reporting project in the Navigator view of BIRT to see the imported report design (.rptdesign) files.
2. Open the first report file in Report Editor by right-clicking the file and then selecting **Open With** → **Report Editor**.
3. Go to the XML source tab.

4. Search for the `odaURL` property and change the its value to point to the local `WLREPORT` directory. For example, if you copied the `WLREPORT` directory to `c:\work\reporting`, then the line in the XML will look like this:

```
<property name="odaURL">jdbc:derby:C:\work\reporting\WLREPORT</property>
```

5. Change the `odaUser` property, located on the next line in the file, to set the user to Worklight, as shown here:

```
<property name="odaUser">Worklight</property>
```

6. Save the file and repeat these steps for each additional report file.

Finally, configure the data source within BIRT:

1. Copy the `derby.jar` file from the Worklight Server (located in the `<wl_install_dir>/server/wlp/usr/shared/resources/derby` directory, where `<wl_install_dir>` is the directory where IBM Worklight Server is installed) to the location where you copied the `WLREPORT` directory in step 2 on page 196 (in this example, `c:\work\reporting`).
2. In the BIRT Report Designer, open the `report_active_users.rptdesign` report file in Report Editor by right-clicking the file and then selecting **Open With** → **Report Editor**.
3. Switch to the Layout tab in the Report Editor, if it is not already shown.
4. Go to the Data Explorer view. If this view is not displayed, select **Window** → **Show View** → **Data Explorer**.
5. On the Data Explorer tab, expand the Data Sources folder (Figure 4-103), right-click Data Source, and then select **Edit**.

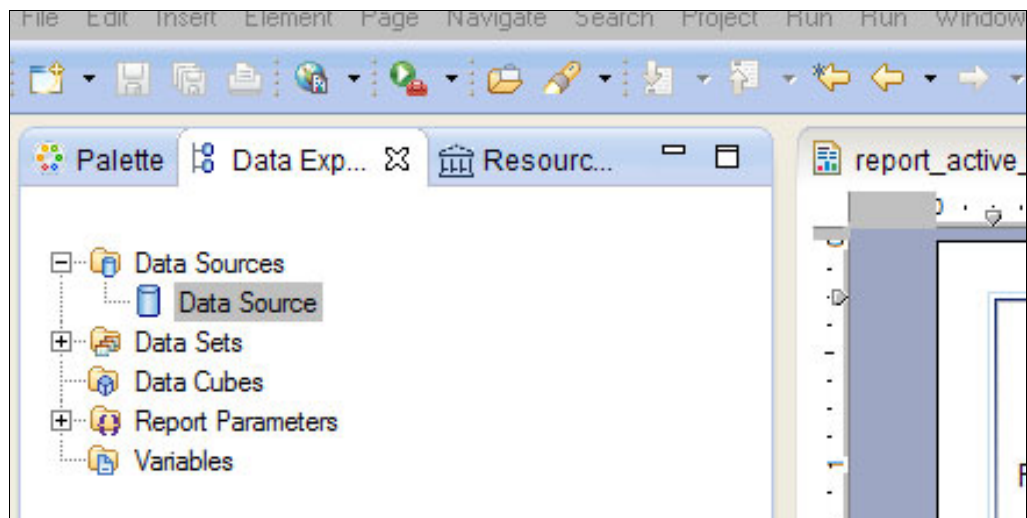


Figure 4-103 The expanded Data Sources folder in the Data Explorer view

6. The data source details are displayed, as shown in Figure 4-104. Click **Manage Drivers** to open the Manage JDBC Drivers dialog.

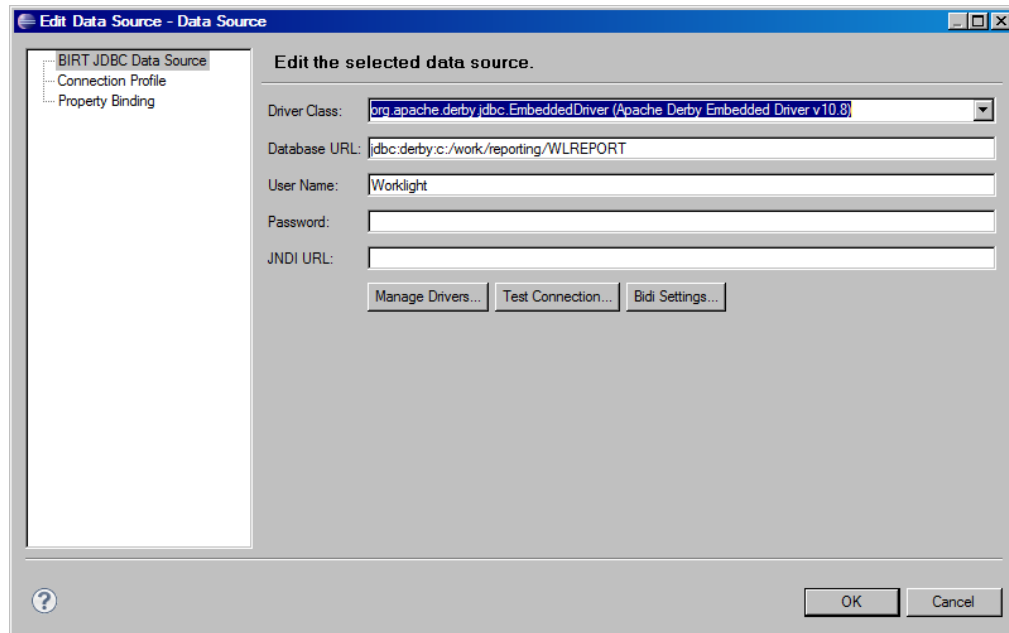


Figure 4-104 Data source details

7. In the Manage JDBC Drivers dialog (shown in Figure 4-105), click **Add**.

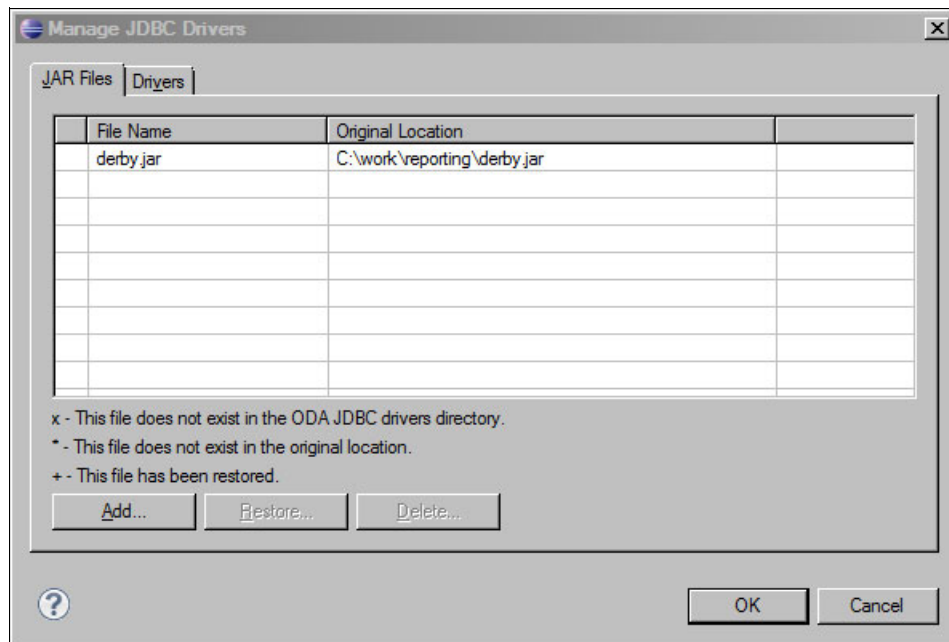


Figure 4-105 Manage JDBC Drivers dialog with the location of the derby.jar file

8. This displays an Open dialog. Navigate to the folder where you placed the copied derby.jar file (in this example, c:\work\reporting), select the derby.jar file, and then click **Open**.

9. The `derby.jar` file and its location on the JAR Files tab of the Manage JDBC Drivers dialog (Figure 4-105 on page 198) is displayed. Click **OK** to save the JDBC driver information and return to the Edit Data Source dialog (Figure 4-104 on page 198).
10. Click **Test Connection** to verify access to the local database using the data source that you just finished configuring.
11. Click **OK** to close the Edit Data Source dialog.

The configuration of the reporting environment is now complete.

4.12.3 Running the reports

You are now ready to use BIRT to run the reports and view the data. These predefined reports that are provided with Worklight Server help you more easily retrieve the valuable information contained within the data that is gathered by the Worklight server, such as determining the number of users accessing the mobile device platform at a specific point in time. With this data, you can monitor the usage of specific mobile applications and use that information as input for future planning.

The predefined BIRT reports that are provided with IBM Worklight Server provide a view into application usage at several levels:

- ▶ Usage of all mobile applications deployed
- ▶ Usage of a specific application, regardless of version
- ▶ Usage of a specific version of an application

In the following sections, several of these reports are run. Depending on the report, the data is shown at the different levels listed previously. Reports that show usage at a specific version of the application will specify a version of the application (such as 1.0) instead of only the application name.

Active user report

The active user report shows the number of active users per day, over a period of 30 days, for all mobile applications hosted on the Worklight server.

To generate the Active user report, go to the Navigator View, right-click the `report_active_users.rptdesign` file, and then select **Report** → **Run Report**.

The report is generated based on the information stored in the WLREPORT database and is displayed in the BIRT Report Viewer, shown in Figure 4-106. The figure shows a total of 15 users on December 18, and a total of 12 users on December 19.

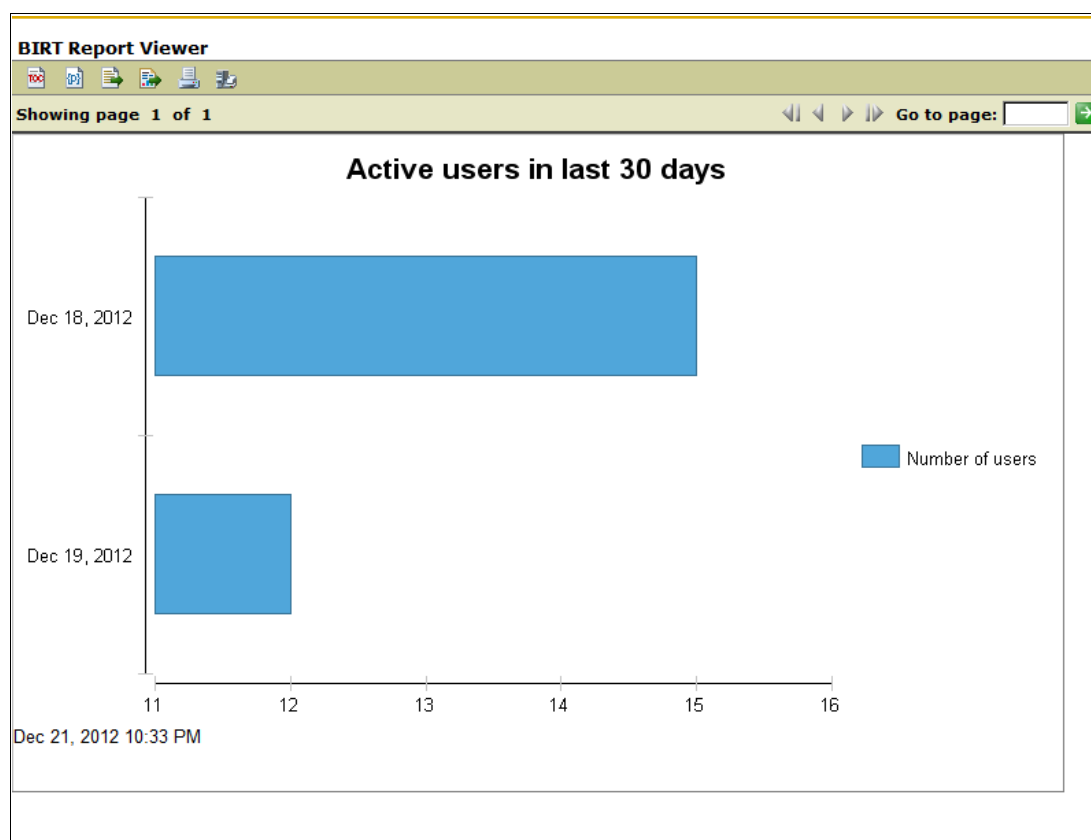


Figure 4-106 Active user report in BIRT Report Viewer

Daily hits report

The Daily hits report shows how often a specific version of the mobile application has been used within the past 30 days. One *hit* is recorded for each request from a mobile application to the Worklight server, such as a call to an adapter to retrieve data. This report provides a picture of the density of communication between the mobile application and the Worklight server and so gives insight as to how much load the application puts on the server.

To generate the Daily hits report, go to the Navigator View, right-click the report_daily_hits.rptdesign file, and then select **Report** → **Run Report**. Because this report requires a parameter, the BIRT Report Viewer is shown and displays a Parameter dialog listing the mobile application versions for which reports can be generated. Select **LuggageTracker 1.0** from the list and then click **OK**.

The report is generated and displayed, as shown in Figure 4-107. As shown in the report, this version of the application had 136 hits (server requests) on December 18 and 140 hits on December 19.

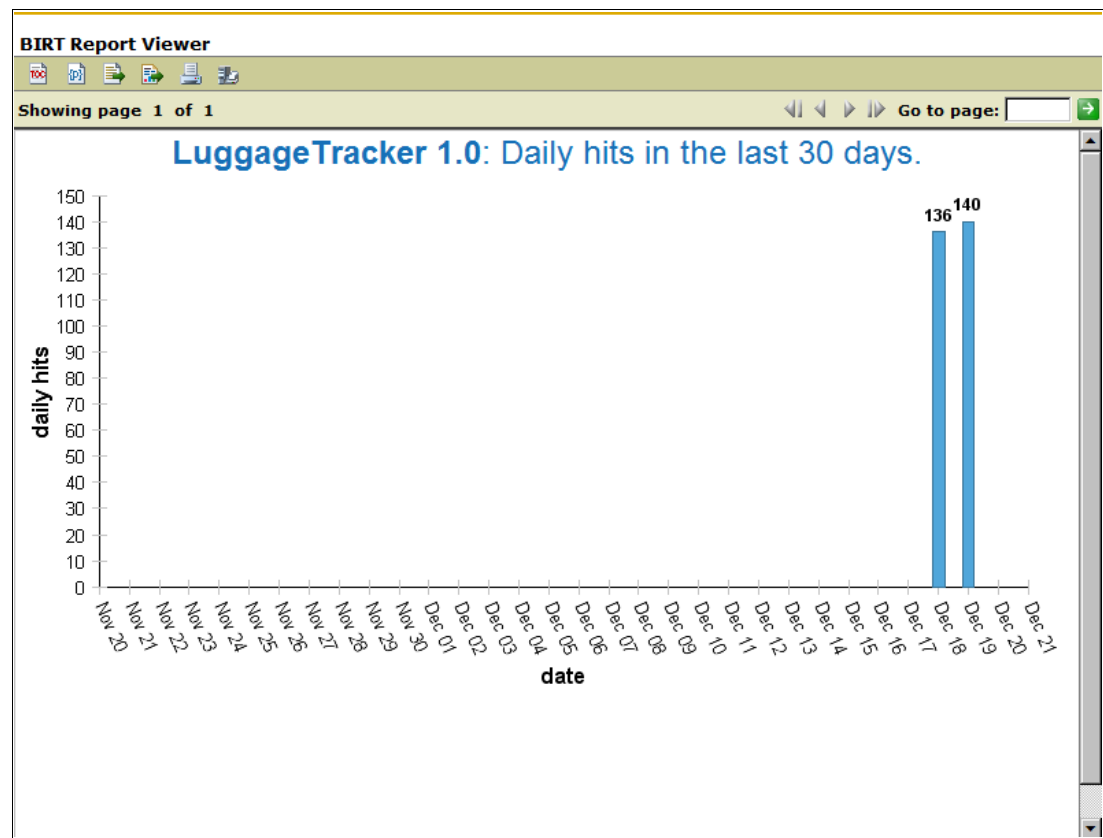


Figure 4-107 Daily hits report for LuggageTracker version 1.0

Daily visits report

The Daily visits report visualizes how often a specific version of a mobile application has been used. One *visit* per day is recorded for each unique request to the Worklight server from a device running a specific application version. So, for example, a user can connect and do three application actions that each cause a request to the server, but this counts as only a single *visit* (even though it created multiple *hits*).

To generate the Daily visits report, go to the Navigator View, right-click the report_daily_visits.rptdesign report file, and then select **Report** → **Run Report**. Because this report requires a parameter, the BIRT Report Viewer is shown and displays a Parameter dialog listing the mobile application versions for which reports can be generated. Select **LuggageTracker 1.0** from the list and then click **OK**.

The report is generated and displayed, as shown in Figure 4-108. As shown in the report, this version of the application was used 9 times on December 18 and 17 times on December 19.

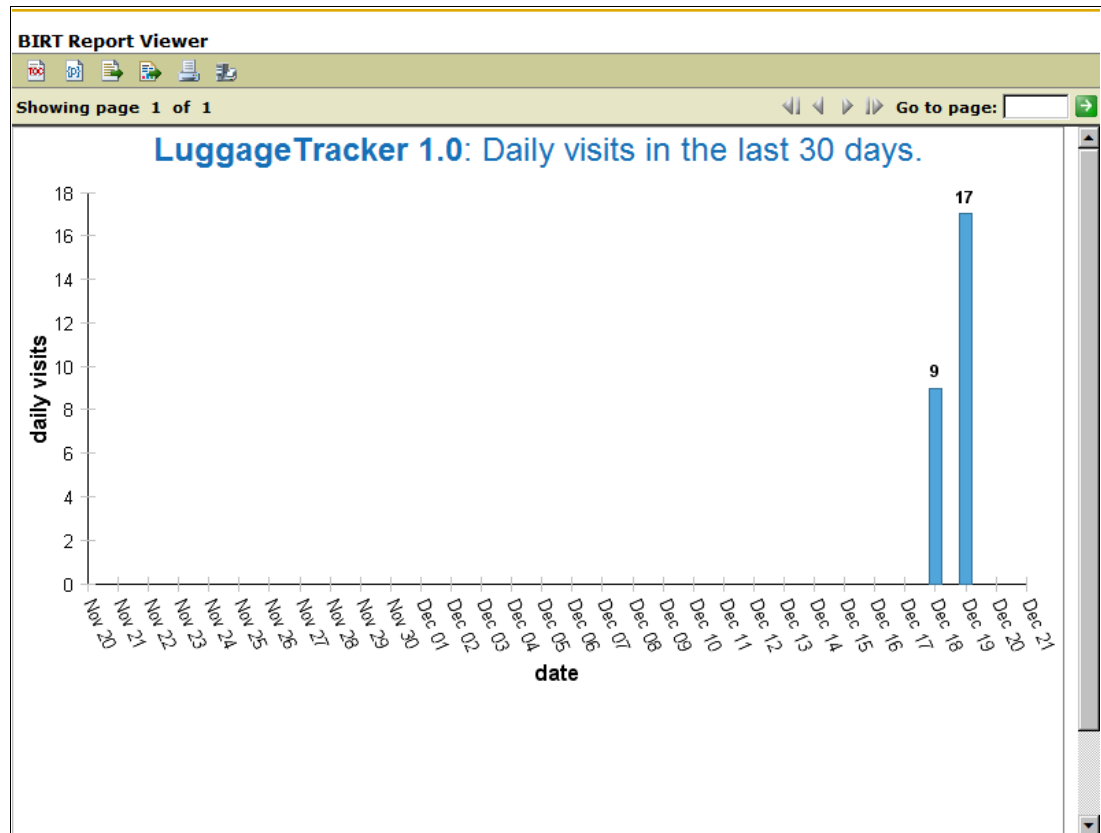


Figure 4-108 Daily visits report for LuggageTracker version 1.0

Environment report

The Environment report visualizes which mobile platforms (such as Android or iOS) have been used to access a specific application.

To generate the Environment report, go to the Navigator View, right-click the report_environment_usage.rptdesign report file, and select **Report** → **Run Report**. Because this report requires a parameter, the BIRT Report Viewer is shown and displays a Parameter dialog listing the mobile applications for which reports can be generated. Select **LuggageTracker** from the list and then click **OK**.

The report is generated and displayed, as shown in Figure 4-109. The report's pie chart shows the visits for each platform per application version (only one version of LuggageTracker was published for each platform). In the table below the chart, you can see that the application was used 20 times on Android devices and 6 times on iOS devices.

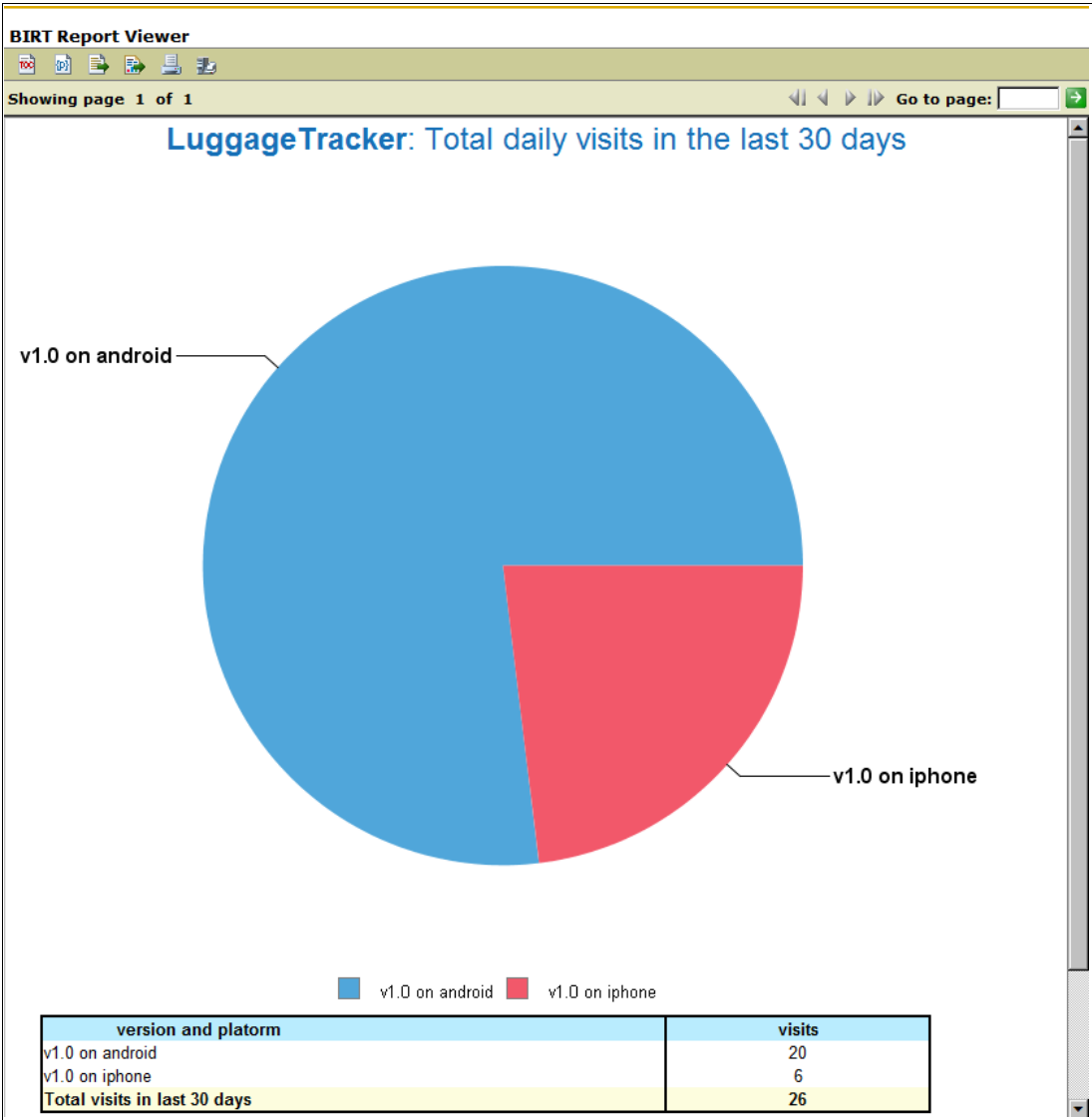


Figure 4-109 Environment report showing breakdown of usage by platform

New devices report

The New devices report summarizes which new devices have been using a specific mobile application in the past 90 days. This report is specifically about devices and not users, because a user can log in to the application from different mobile devices.

To generate the New devices report, go to the Navigator view, right-click the report_new_devices.rptdesign report file, and then select **Report** → **Run Report**. Because this report requires parameters, the BIRT Report Viewer is shown and displays a Parameter dialog listing the mobile applications for which reports can be generated. Select **LuggageTracker** from the list and click **OK**.

The report is generated and displayed, as shown in Figure 4-110. The report shows the number of new devices that were detected in the last 90 days and for each one, lists the Device ID and the first day it was used to access LuggageTracker.

BIRT Report Viewer		
Showing page 1 of 1		
Go to page: <input type="text"/>		
LuggageTracker: New devices detected in the last 90 days.		
New Device Count: 13		
Device Details		
#	DEVICE ID	RECORDED DATE
1	35b69306-f033-486e-ae75-a06b2d6d6379	Dec 18, 2012 12:00 AM
2	491d9d32-081c-4627-ae7d-640bf2940606	Dec 18, 2012 12:00 AM
3	4c2e8bb6-f2d6-49b3-8e18-9c12593fd062	Dec 18, 2012 12:00 AM
4	4f4845cb-5969-4486-bd63-42d1da79e5c6	Dec 18, 2012 12:00 AM
5	540dd67a-bea6-4de8-ba1b-c5f2469a108a	Dec 18, 2012 12:00 AM
6	68417e09-c51a-4d6c-bcb1-62b9422775a2	Dec 18, 2012 12:00 AM
7	9362006e-f615-46bb-9148-7a517f749f5b	Dec 19, 2012 12:00 AM
8	C5ACD90C-8497-4BDE-ABD0-2EA744EB4892	Dec 19, 2012 12:00 AM
9	F32C165B-2AB1-4847-B88F-F968B1BA8645	Dec 19, 2012 12:00 AM
10	c2808e3d-b334-49a7-b4f2-2f11497e7cd9	Dec 19, 2012 12:00 AM
11	c8185d71-e0b1-4882-a067-afc96392105c	Dec 18, 2012 12:00 AM
12	d4bdc538-472d-4f7e-a427-5d4d812a8b37	Dec 19, 2012 12:00 AM
13	f4910597-a3f3-4ffc-b008-d06fcc418faa	Dec 19, 2012 12:00 AM
Dec 21, 2012 11:00 PM		

Figure 4-110 New device report for LuggageTracker

Expanding the solution to consumers

This chapter describes the second phase of the mobile luggage tracking solution for Airline Company A, which expands the application to the airline's primary consumers, its passengers. It builds upon the information provided in Chapter 4, "Creating and deploying the mobile solution" on page 59, where an initial application was built for use by airline customer service representatives, and introduces these new features:

- ▶ Integration with IBM WebSphere eXtreme Scale
- ▶ Integration with the IBM DataPower XG45 appliance
- ▶ Integration with WebSphere Cast Iron

The following topics are covered:

- ▶ 5.1, "Breaking down the new scenario" on page 207
- ▶ 5.2, "Changes to the environments" on page 215
- ▶ 5.3, "Changes to the back-end services" on page 232
- ▶ 5.4, "Creating the new mobile application" on page 239
- ▶ 5.5, "Unit testing" on page 290
- ▶ 5.6, "Deploying the application" on page 291

The chapter includes snippets of source code used in the mobile solution and assumes that readers understand basic application development principles. It also assumes knowledge of Eclipse and its functions. The iOS-specific sections assumes knowledge of the Xcode IDE.

IBM DOES NOT WARRANT OR REPRESENT THAT THE CODE PROVIDED IS COMPLETE OR UP-TO-DATE. IBM DOES NOT WARRANT, REPRESENT OR IMPLY RELIABILITY, SERVICEABILITY OR FUNCTION OF THE CODE. IBM IS UNDER NO OBLIGATION TO UPDATE CONTENT NOR PROVIDE FURTHER SUPPORT.

ALL CODE IS PROVIDED "AS IS," WITH NO WARRANTIES OR GUARANTEES WHATSOEVER. IBM EXPRESSLY DISCLAIMS TO THE FULLEST EXTENT PERMITTED BY LAW ALL EXPRESS, IMPLIED, STATUTORY AND OTHER WARRANTIES, GUARANTEES, OR REPRESENTATIONS, INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE,

AND NON-INFRINGEMENT OF PROPRIETARY AND INTELLECTUAL PROPERTY RIGHTS. YOU UNDERSTAND AND AGREE THAT YOU USE THESE MATERIALS, INFORMATION, PRODUCTS, SOFTWARE, PROGRAMS, AND SERVICES, AT YOUR OWN DISCRETION AND RISK AND THAT YOU WILL BE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY RESULT, INCLUDING LOSS OF DATA OR DAMAGE TO YOUR COMPUTER SYSTEM.

IN NO EVENT WILL IBM BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY OR CONSEQUENTIAL DAMAGES OF ANY TYPE WHATSOEVER RELATED TO OR ARISING FROM USE OF THE CODE FOUND HEREIN, WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, LOST SAVINGS, LOSS OF PROGRAMS OR OTHER DATA, EVEN IF IBM IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS EXCLUSION AND WAIVER OF LIABILITY APPLIES TO ALL CAUSES OF ACTION, WHETHER BASED ON CONTRACT, WARRANTY, TORT OR ANY OTHER LEGAL THEORIES.

THIRD PARTY SOFTWARE IS LICENSED AND DISTRIBUTED TO YOU BY THE THIRD PARTY DISTRIBUTORS AND/OR RESPECTIVE COPYRIGHT AND OTHER RIGHT HOLDERS UNDER THEIR TERMS AND CONDITIONS. IBM MAKES NO EXPRESS OR IMPLIED WARRANTIES OR REPRESENTATIONS WITH RESPECT TO SUCH SOFTWARE AND PROVIDES NO INDEMNITY FOR SUCH SOFTWARE. IBM GRANTS NO EXPRESS OR IMPLIED PATENT OR OTHER LICENSE WITH RESPECT TO AND IS NOT LIABLE FOR ANY DAMAGES ARISING OUT OF THE USE OF SUCH SOFTWARE.

5.1 Breaking down the new scenario

Because of positive feedback from customers and CSRs following the release of the initial LuggageTracker application, Airline Company A has decided to proceed with the next phase of its mobile solution roadmap. In this phase, a second application called *MyLuggage* will be created to give airline passengers access to the same kind of luggage-tracking information that CSRs can get using LuggageTracker.

In contrast to LuggageTracker, the first version of MyLuggage will only support Android devices.

5.1.1 High-level use case

The high-level use case for the internal luggage tracking solution (described in 4.1.1, “High-level use case” on page 61) also applies to the new, business-to-consumer application, which will provide a similar set of capabilities but to a different set of users. In its first release, MyLuggage will support users who have registered to use the airline’s website and therefore already have a username and password stored in the airline’s back-end systems. These same credentials will be required to access the new mobile application.

5.1.2 Agile breakdown and design

Using agile methodologies, the high-level use case is broken down into four main scenarios or *user stories*:

- ▶ **Install:** As a passenger with a supported mobile device, I need to find MyLuggage in my device vendor’s application store and then install it using the mechanism provided.
- ▶ **Login:** As a registered user of the airline’s website, I need to launch the new MyLuggage application I just installed and then log in using my existing credentials.
- ▶ **Locate:** As a logged-in user, I need to locate my checked bag when it does not arrive at my destination as expected.
- ▶ **Deliver:** As a registered user with a delayed bag that must be delivered to me later, I need to update my baggage record with a delivery address, either by entering it manually or by selecting it from a map.

For each of these user stories, the airline’s mobile development and design teams completed a storyboard to list the specific interactions between the user, the mobile application, and the airline’s back-end services. These storyboard mock-ups or *flows* are shown in the following subsections.

From a development perspective, each story is broken down into tasks (see 5.1.3, “Solution components” on page 211) and each task is assigned to appropriate IT staff (see 5.1.5, “New tasks to implement the customer solution” on page 214). This task breakdown also involves completing the low-level application designs and expanding the airline’s mobile solution component model to meet the needs of the new application.

Install flow

The storyboard mock-up of the Install user story is shown in Figure 5-1. It details the initial installation of the application on the mobile device and is a “one-time story” in that the user installs the application only once.

The steps within the Install flow are as follows:

1. A mobile device user searches the device vendor’s application store for the MyLuggage application.
2. The user clicks **Install** to download and install the application on their device.

After the application is installed, it can be started using the application icon on the mobile device screen.

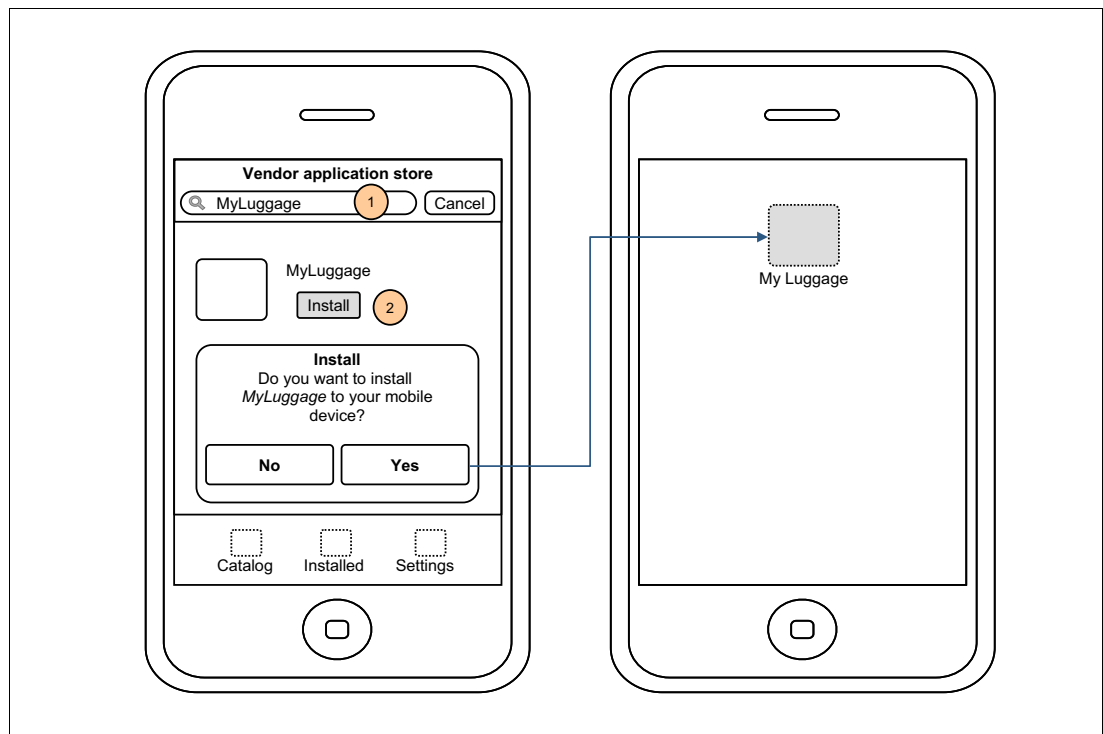


Figure 5-1 Storyboard of Install user story

Login flow

The storyboard mock-up of the Login user story is shown in Figure 5-2. It assumes that the Install user story has been completed.

The steps in the Login flow are as follows:

1. The user starts MyLuggage by clicking its icon on the mobile device screen.
2. A login dialog prompts for authentication information. The user provides the same username and password that the user previously created to use the airline's website.
3. The user's credentials are authenticated against the airline's back-end user registry.

Upon a successful login, the MyLuggage home screen is displayed with these options:

- Scan Bar Code
- Retrieve Information
- Logout

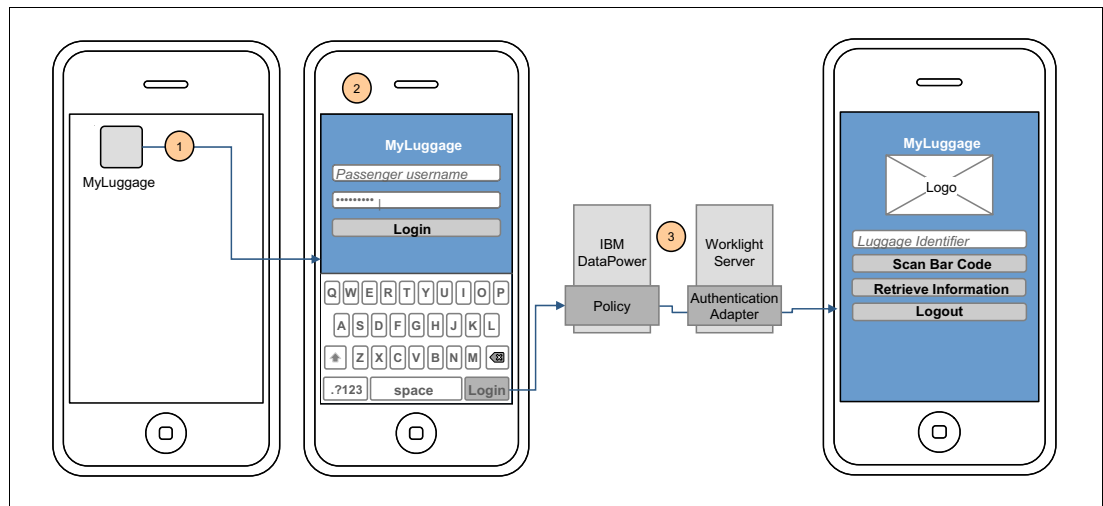


Figure 5-2 Storyboard of Login user story

Locate flow

The storyboard mock-up of the Locate user story is shown in Figure 5-3. It assumes that the Login user story has been completed.

The steps in the Locate flow are as follows:

1. The user clicks **Scan Bar Code** and points the mobile device's camera at their luggage receipt.
2. The application scans the bar code on the luggage receipt and displays the luggage identifier number on the device screen. If scanning fails, the user uses the device keyboard to manually enter the luggage identifier number into the application interface.
3. With the luggage identifier information successfully scanned or entered, the user clicks **Retrieve Information**.
4. Information about the bag is retrieved from the airline's back-end systems, specifically, the Luggage Tracking Information System (LTIS), as follows:
 - Owner's name
 - Current luggage status (such as Delayed or On-time)
 - Current, next, and final destination
 - Date of last status update
 - Comments previously entered by the passenger (if any)

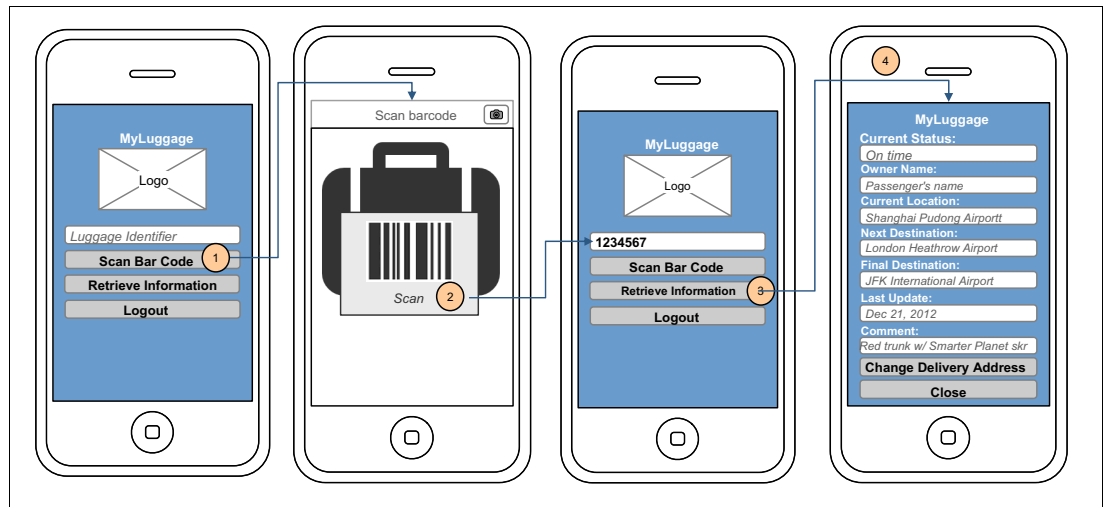


Figure 5-3 Storyboard of Locate user story

Deliver flow

The storyboard mock-up of the Deliver user story is shown in Figure 5-4. It assumes that the Locate user story has been completed and that the passenger's delivery address for delayed bags must be updated.

The steps in the Deliver flow are as follows:

1. The user clicks **Change Delivery Address** to display a screen in which to enter a preferred delivery address, phone number, and comments.
2. The user enters the delivery address with one of two methods:
 - By touching the appropriate text field and then typing the address using the device keyboard
 - By clicking **Map** to view a map on which they can designate the address by touching its location on the display
3. The user enters a phone number and adds any comments, such as the a preferred delivery time, into the Comments field.
4. The user clicks **Save and return** to save the information in the LTIS and redisplay the Luggage information screen.

The LTIS record for the bag is now updated and can be used by the delivery service assigned to deliver late-arriving bags to passengers.

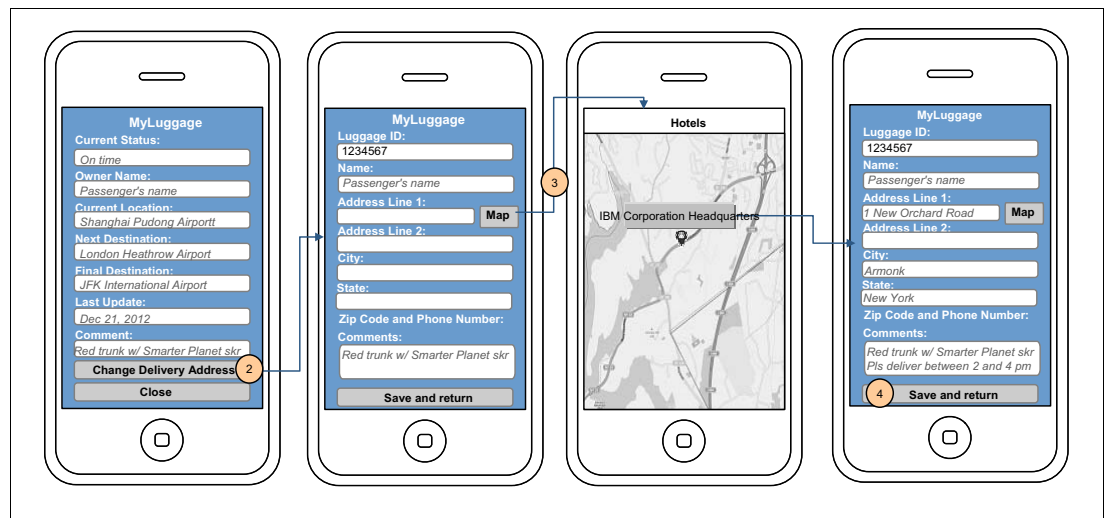


Figure 5-4 Storyboard of Deliver user story

5.1.3 Solution components

The airline's mobile luggage tracking solutions, both the internal application and the new customer-facing application, are supported by the same component model. The original model for LuggageTracker contained three layers (see in 4.1.3, "Components of mobile luggage tracking solution" on page 65). Now, however, a fourth layer must be added for a secure reverse proxy server to handle user authentication. The secure reverse proxy server handles the authentication that was done by the authentication adapter in the internal application and the new layer for it exists between the mobile application layer and the mobile middleware layer.

Within each layer of the model is a set of core components, as shown in Figure 5-5. In the figure, the components with dashed borders are components of IBM Worklight Server; those with double-line borders are custom components written by the airline's IT department.

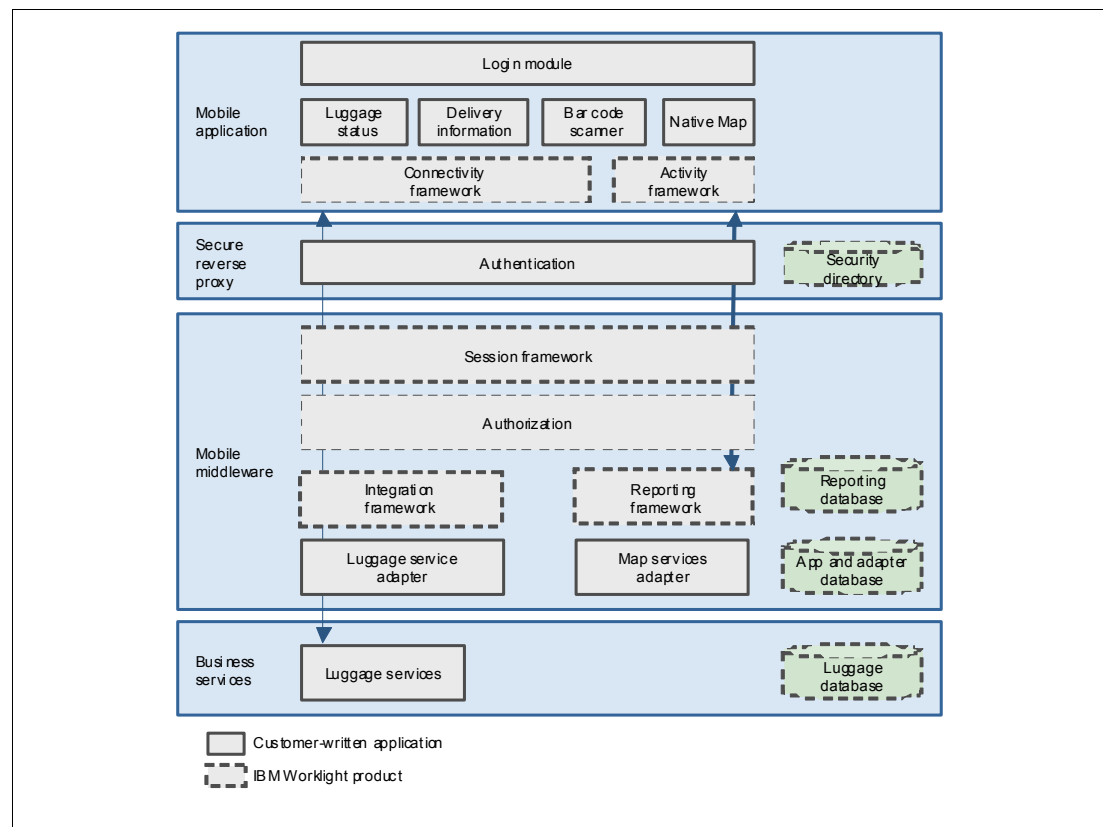


Figure 5-5 Component model including new, passenger-facing application

To integrate the map-based address feature, the updated component model also includes a new native map component in the mobile application layer and a Map services adapter in the mobile middleware layer.

5.1.4 Expanding the topology

The MyLuggage application offers the same functions that are available to CSRs using LuggageTracker. Yet expanding the user base to include passengers complicates certain aspects of the solution topology, which must be expanded to meet these additional needs:

- ▶ Authentication and authorization of passengers through the use of existing back-end systems and registration data
- ▶ Minimization of third-party costs for services such as retrieving lists of hotels that are near to a specific geographic location.
- ▶ Integration with device-based location services, such as the mapping tools that will be used to designate a passenger's preferred delivery location

To reduce development and maintenance costs, the new application must reuse the existing infrastructure and components of LuggageTracker when possible. The expanded topology for the passenger-facing application is shown in Figure 5-6.

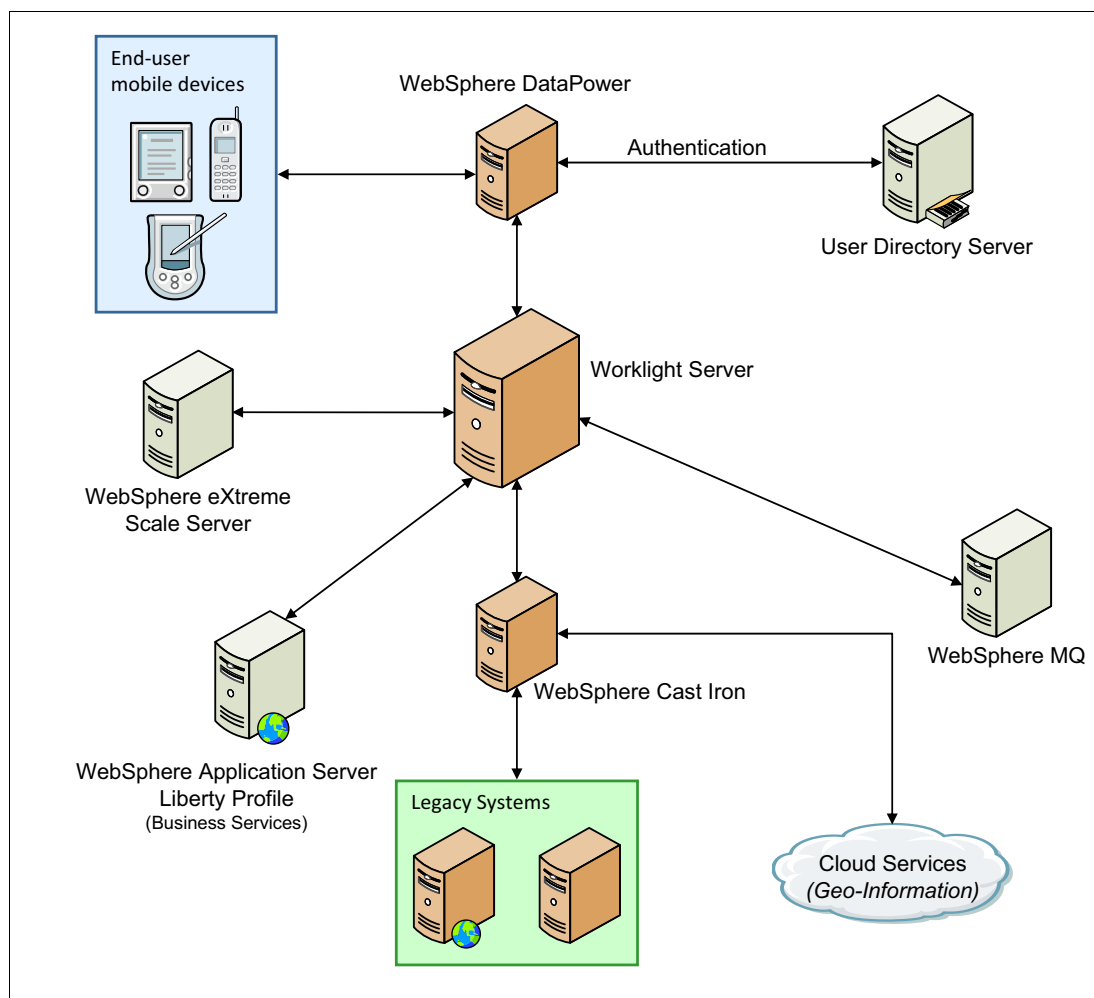


Figure 5-6 Topology including new passenger-facing application

Authentication and authorization

In contrast to LuggageTracker, MyLuggage is a business-to-consumer application and therefore requires enhanced security mechanisms. Making the IBM Worklight server accessible to passenger-owned devices opens Airline Company A's IT infrastructure to inbound connections, which are potential security exposures if not controlled properly. Therefore, a secure gateway must be introduced to protect the existing IT infrastructure and restrict access to only those back-end services that are required by the new application. The IBM WebSphere DataPower XG45 appliance will provide firewall capabilities to protect internal systems. Using this appliance as the secure gateway creates what is essentially a self-contained security infrastructure for the new application.

Minimizing costs of third-party services

If third-party services, such as mapping services, will impose costs to the airline on a per-request basis, these costs can be minimized by caching the results of calls that are frequently repeated. Many users look up the same hotel, so caching this hotel information using IBM WebSphere eXtreme Scale not only reduces the per-request charges, it improves performance by replacing repeated calls to the external service with simpler, shorter calls to

the internal caching system. The number of network hops decreases, which can result in reduced latency. And because WebSphere eXtreme Scale is a software solution, it requires no special hardware and can be installed on existing servers.

Integrating with location services

Ensuring that the passenger-supplied delivery address is valid requires additional data quality checks. The MyLuggage application uses the mobile device's onboard location service to determine the passenger's current location and then retrieves a list of nearby hotels from a third-party service. By allowing the user to select one of these hotels, the accuracy of the delivery address is improved and the potential for errors based on manually input data is reduced. Because Airline Company A lacks its own database of worldwide hotel locations, the application accesses an external, third-party service using IBM WebSphere Cast Iron.

5.1.5 New tasks to implement the customer solution

The MyLuggage application will reuse many of the existing back-end services that were created for LuggageTracker, but will require a new user interface and security mechanism. The following list describes the tasks that are required for creating these new features by each involved member of the airline's IT department:

- ▶ The new application user interface is created by a *web mobile application developer* using IBM Worklight Studio. To demonstrate how Worklight Studio supports other frameworks besides jQuery, the new mobile application is implemented using Dojo Mobile. This is done only to show the additional framework support available in Worklight Studio; there is no advantage in using Dojo Mobile instead of jQuery in this scenario.
- ▶ To support the selection of an address using device-specific mapping tools requires collaboration by these individuals:
 - An *integration developer* uses IBM WebSphere Cast Iron to gather hotel information from a third-party service.
 - The list of hotels from which the passenger can select is displayed on an interactive map using native device functionality. A *native mobile application developer* creates the native map page and the function to display the nearby hotels.
 - A *mobile server developer* adds caching of the third-party hotel information using WebSphere eXtreme Scale.
- ▶ To add the required security mechanism, a *system administrator* integrates IBM Worklight Server with an IBM WebSphere DataPower XG45 appliance to connect the application with the existing security infrastructure and credentials database.

The remaining sections in this chapter explain how Airline Company A implemented the new passenger-facing application and provide step-by-step instructions for accomplishing the required tasks. If you want to implement the end-to-end scenario, you must complete each section.

5.2 Changes to the environments

Several new hardware and software products must be added to Airline Company A's pre-production environment. In addition, the development environment requires updates to allow the native mobile application developer to add the native map functionality to MyLuggage.

The following subsections explain the changes to the development and pre-production environments.

Important: For the work described in this chapter, use the same IBM Worklight Studio workspace that you used for Chapter 4, "Creating and deploying the mobile solution" on page 59. You can switch workspaces in Worklight Studio by selecting **File** → **Switch Workspace** and then selecting the desired workspace.

5.2.1 Updating the development environment

With the new features that will be included in MyLuggage, there are several additions to the development environment for native mobile application developers, web mobile application developers and mobile server developers.

Native mobile application developer

As part of the delivery address-related functions in the new MyLuggage application, which will be available only for Android devices, a native Google Map page will be displayed. To enable this display, the native mobile application developer must do these tasks:

- ▶ Obtain a Google Maps API Key
- ▶ Update the Worklight Studio development environment to add Google Play services

The Google Maps API Key and Google Play Services are used by the native mobile application developer in 5.4.2, "Developing the native map page" on page 264 when the native map function is created.

Obtaining a Google Maps API key

The Google Maps API key is provided by Google and enables you to monitor your application's Maps API usage, and ensures that Google can contact you about your application if necessary. There are usage limits for the free key and if you exceed this quota, you must purchase additional quota from Google. To request a key, you must supply the package name of the Android application (com.AirlineShellComponentTest in this scenario) and the SHA-1 key of your signing certificate.

For details about obtaining a Google Maps API Key and retrieving the SHA-1 key of your certificate, see the Google Developers website at this address:

https://developers.google.com/maps/documentation/android/start#getting_the_google_maps_android_api_v2

Adding Google Play services

To add the Google Play services library project to IBM Worklight Studio, install the Google Play services SDK with Android SDK Manager as follows:

1. Start IBM Worklight Studio using the same workspace that was used in Chapter 4, "Creating and deploying the mobile solution" on page 59.
2. Click **Window** → **Android SDK Manager** to display Android SDK Manager.

3. Deselect any updates for installed packages, if any of them are pre-selected.
4. Expand the Extras folder and select **Google Play services**, as shown in Figure 5-7.

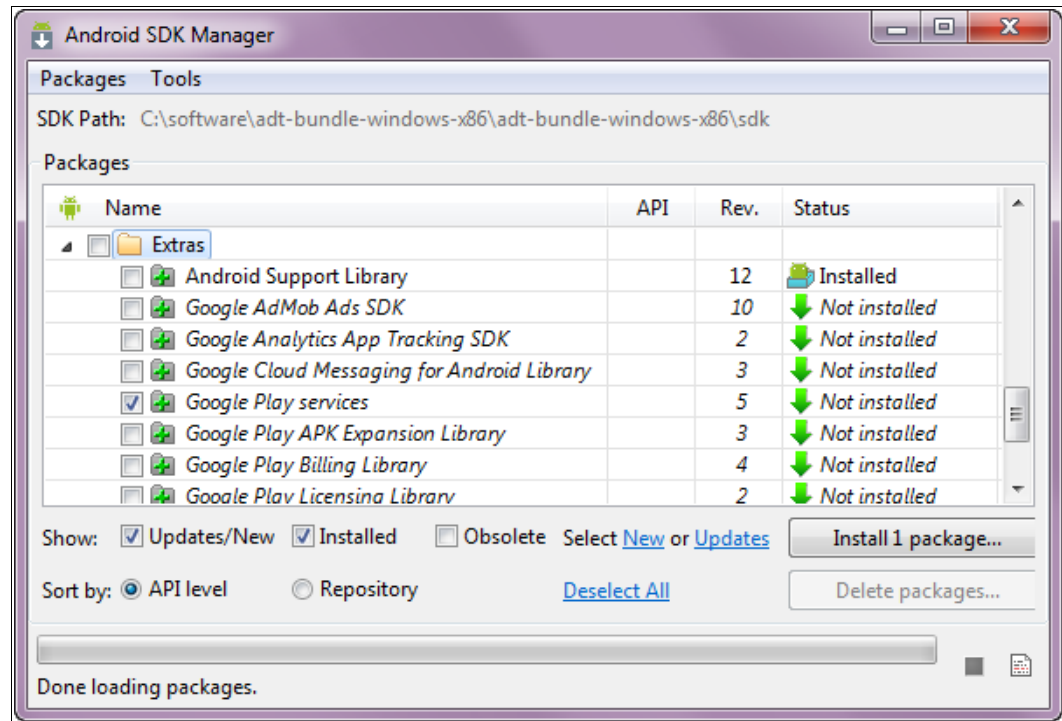


Figure 5-7 Selecting Google Play services from Android SDK Manager

5. Click **Install 1 package**.
6. Read and accept the software license information and then click **Install**. As the Google Play services are downloaded and installed, you can follow the progress at the bottom of the dialog.
7. When installation completes, close Android SDK Manager.

Now that Google Play services are available for use in Worklight Studio, you must import the Google Play services library project into your workspace:

1. In Worklight Studio, click **File** → **Import**.
2. In the Import wizard, expand the android folder and select **Existing Android Code Into Workspace**, as shown in Figure 5-8, and then click **Next**.

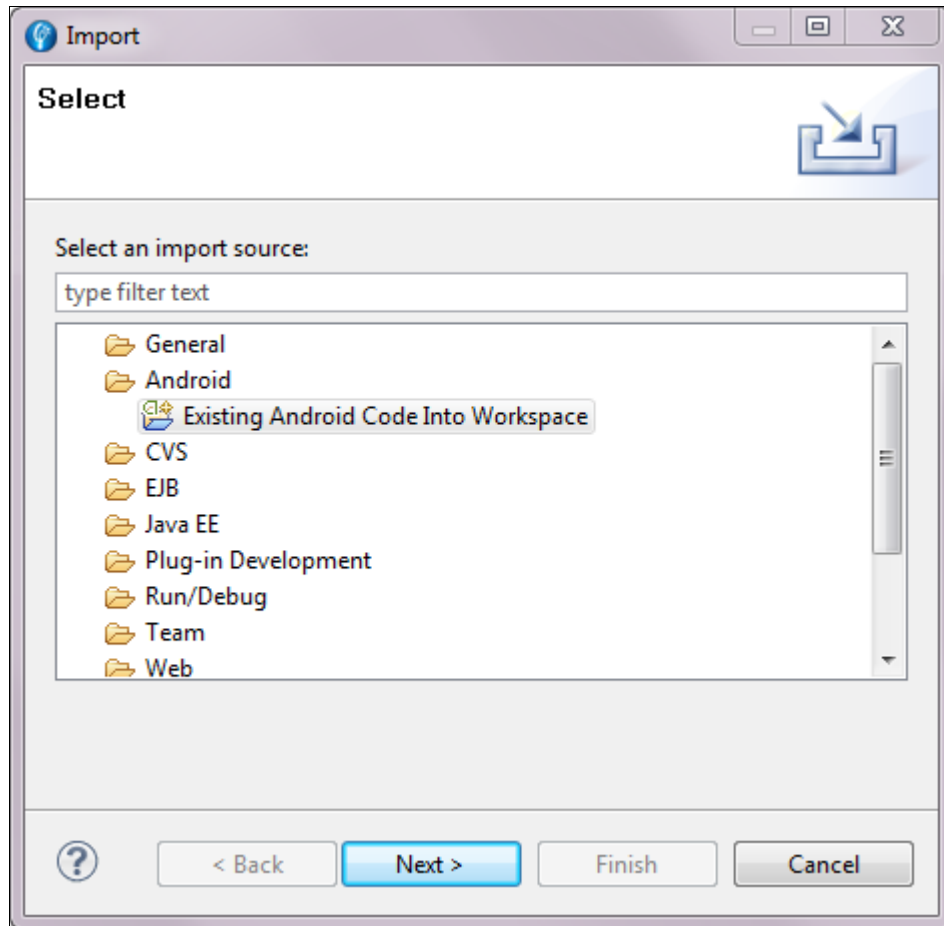


Figure 5-8 Selecting to import existing Android code into the workspace

3. The Import Projects page of the wizard is displayed (Figure 5-9 on page 218). Click **Browse** to display the Browse For Folder dialog and locate the directory of the Google Play services library project (where `<sdk_path>` is the path for the Android SDK):

`<sdk_path>/extras/google/google_play_services/libproject`

Select the `libproject` directory and click **OK**.

HINT: If you do not know what to use for `<sdk_path>`, open Android SDK Manager and you will see the SDK Path listed at the top just below the menu bar.

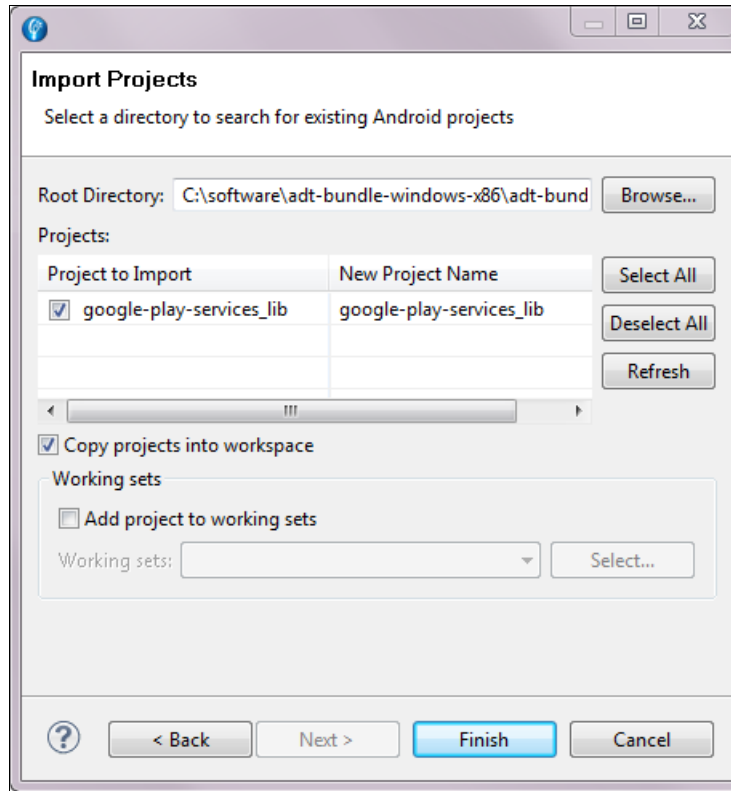


Figure 5-9 Copying the Google Play Services project into the workspace

4. The directory of your `google-play-services_lib` project is now in the Root Directory field of Import Projects page. Ensure that the check box next to `google-play-services_lib` in the Project To Import column of the table checked, select the **Copy projects into workspace** check box and then click **Finish**.

You now have a new project in Project Explorer called `google-play-services_lib`.

Important: An Android application using the Google Play services library project can be tested only on a real device and not with the Android emulator. For information about this limitation, see the Google Developers Setup documentation for Google Play Services:

<http://developer.android.com/google/play-services/setup.html>

For further information regarding the use of Google Maps in an Android-based mobile application, see the Google Developers documentation:

<https://developers.google.com/maps/documentation/android>

Web mobile application developer

The security mechanism for MyLuggage will use the IBM WebSphere DataPower XG45 appliance and its use in the mobile application requires a base 64 encoding package. Therefore, the mobile application developers must add the selected package to their development environment.

Adding a base 64 encoding package

The IBM DataPower XG45 appliance expects the HTTP authorization header to be encoded to a base 64 file format.

To encode and decode the authorization header, the client-side authentication uses a third-party JavaScript library. Various base 64 encoding and decoding libraries can be used. For the purpose of this book, the authors used a JavaScript library from webtoolkit, called *Javascript base64*, available at this address:

<http://www.webtoolkit.info/javascript-base64.html>

To use the webtoolkit JavaScript base64 library in the development environment, copy the `webtoolkit.base64.js` file into the `apps/common/js` folder of the `LuggageTrackerDojo` application, naming it `base64.js`. If multiple developers are collaborating to develop the application, only one developer needs to add this file into the source code repository.

Mobile server developers

Caching functionality is being added to the adapters using WebSphere eXtreme Scale. As part of this function, the development environment must be updated with the following items:

- ▶ A Java-based JavaScript implementation
- ▶ The WebSphere eXtreme Scale client

Adding a Java-based JavaScript implementation

The hotel adapter that will be created for the new mapping function will use a third-party Java-based JavaScript implementation to provide scripting capabilities to the server-side caching classes.

For this scenario, the Mozilla Rhino third-party library is used to handle JavaScript objects that sent to and from Java classes. The Mozilla Rhino project is available in most Eclipse installations. To use it, copy a `js.jar` file, such as the one from your Worklight Studio installation, into the `server/lib` folder of the `AirlineAdapterProject` project. If multiple developers are collaborating to develop the application, only one developer needs to add this file into the source code repository.

Adding the WebSphere eXtreme Scale client

The hotel adapter will be updated to cache the hotel information that it obtains from the third-party service using WebSphere eXtreme Scale. To develop and test the caching functions, the WebSphere eXtreme Scale client API JAR file needs to be added to their development environment.

For this scenario, copy the WebSphere eXtreme Scale client API JAR file named `osclient.jar` (located in the `<wxs_install_dir>/ObjectGrid/lib` directory of the WebSphere eXtreme Scale installation) into the `server/lib` folder of the `AirlineAdapterProject` project in Worklight Studio. If multiple developers are collaborating to develop the application, only one developer needs to add this file into the source code repository.

Troubleshooting: At the time of writing, the authors found a conflict in the OSGi framework in which the WebSphere eXtreme Scale client API jar (`ogclient.jar`) cannot be added, as-is, to IBM Worklight Server. To resolve this conflict, extract the contents of the jar into a temporary location, remove the `osgi` folder, and then re-package the jar file.

5.2.2 Changes to the pre-production environment

To support the new features in MyLuggage, the pre-production environment must be expanded to include these tasks:

- ▶ Installing and configuring WebSphere eXtreme Scale for caching
- ▶ Installing and configuring IBM WebSphere Cast Iron
- ▶ Configuring the IBM DataPower XG45 appliance

Installing and configuring WebSphere eXtreme Scale

IBM WebSphere eXtreme Scale is the caching solution used for Airline Company A's business-to-consumer application. If you do not already have WebSphere eXtreme Scale installed, see 6.6, "Installing IBM WebSphere eXtreme Scale" on page 319 for installation details.

This scenario uses a simple, single-grid configuration that is included in the Getting Started sample application. This sample application is part of the default product installation and can be found on the server where IBM WebSphere eXtreme Scale is installed in the following directory (<wxs_install_root> is the directory where the product was installed):

<wxs_install_root>/ObjectGrid/gettingstarted

This scenario uses the configuration from the `objectgrid.xml` file in the <wxs_install_root>/ObjectGrid/gettingstarted/xml directory. This file defines a single grid, named Grid, and two maps, named Map1 and Map2. No additional changes are necessary in this scenario.

To start the grid that will be used in this scenario, use the following steps:

1. Open a terminal session on the IBM WebSphere eXtreme Scale server, using an ID and password for a privileged user.
2. Change to the <wxs_install_root>/ObjectGrid/gettingstarted directory.
3. Start a catalog services process by running the `./runcat.sh` command.
4. Open another command window and change to the same gettingstarted directory.
5. Run a container service instance by running the `./runcontainer.sh server0` command.

The catalog and container services are now running and able to process caching requests.

For additional information about this sample, see the `GETTINGSTARTED_README.html` file located in the <wxs_install_root>/ObjectGrid/gettingstarted directory.

Installing and configuring WebSphere Cast Iron

In this scenario, IBM WebSphere Cast Iron is used for retrieving hotel information from a third-party web service that takes specified a geographic location and returns a list of nearby hotels. This is used in the MyLuggage application to help users enter the address of a hotel for baggage delivery.

To install and configure WebSphere Cast Iron, several tasks must be completed:

- ▶ Install WebSphere Cast Iron
- ▶ Register for the third-party services
- ▶ Create the Cast Iron orchestration to retrieve the information from the third-party service

Installing WebSphere Cast Iron

To install and configure WebSphere Cast Iron for this scenario, see 6.5, "Installing IBM WebSphere Cast Iron" on page 315.

Registering for the third-party services

In this scenario, the Nokia Places API is used to retrieve hotel information and show how a solution can use third-party service providers. For further information about Nokia HERE and the Nokia Places API, see the documentation at this address:

<http://developer.here.com/docs/places/>

To use the Nokia Places API, you must register with Nokia and receive credentials. These credentials, called an *App Token*, consist of an application identifier (app_id) and an application code (app_code). To register and receive your credentials, see the information at this address:

<http://developer.here.com/docs/places/#common/credentials.html>

Creating the orchestration

The step-by-step creation of the Cast Iron orchestration for retrieving the hotel information will not be covered in detail here. Instead, the exported WebSphere Cast Iron project (NokiaHere.par) and the project source code (NokiaHere.zip) are provided as additional material for this book. Only the exported project file is required.

After you download the exported project file and obtain the needed credentials from Nokia, deploy the project as follows:

1. Start the WebSphere Cast Iron web management console as shown in Figure 5-10, in a browser using the following address (where `<castiron_admin_ip_address>` is the admin IP address for your Cast Iron installation):

`https://<castiron_admin_ip_address>/login`

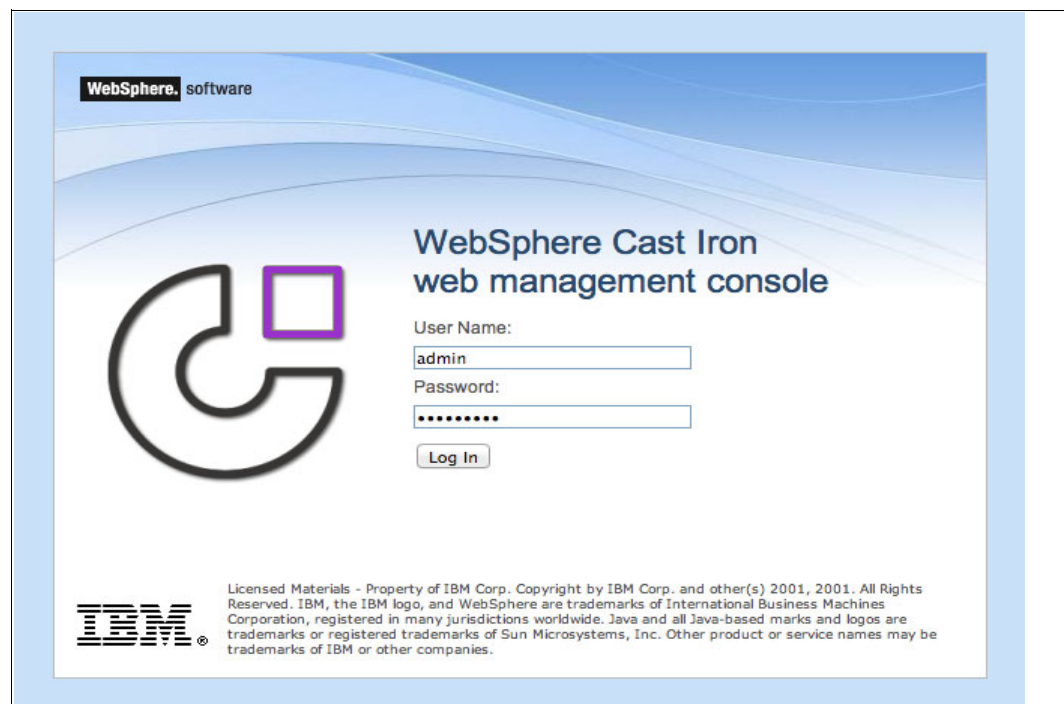


Figure 5-10 Logging into the WebSphere Cast Iron web management console

2. Log in as `admin` using the proper admin password (by default, it is `!n0r1t5@C`).

3. In the navigation pane, click **Repository** to go to the repository view so you can upload a new project, as shown in Figure 5-11.

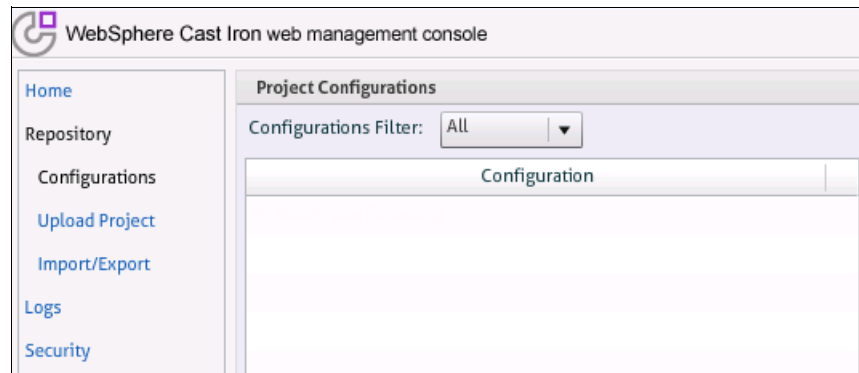


Figure 5-11 Changing to the Repository view

4. Click **Upload Project** to display the Upload Project Configuration dialog,
5. Click **Browse** and select the WebSphere Cast Iron project file (NokiaHere.par) that you downloaded previously. The file name is displayed in the Upload Project configuration dialog, shown in Figure 5-12.

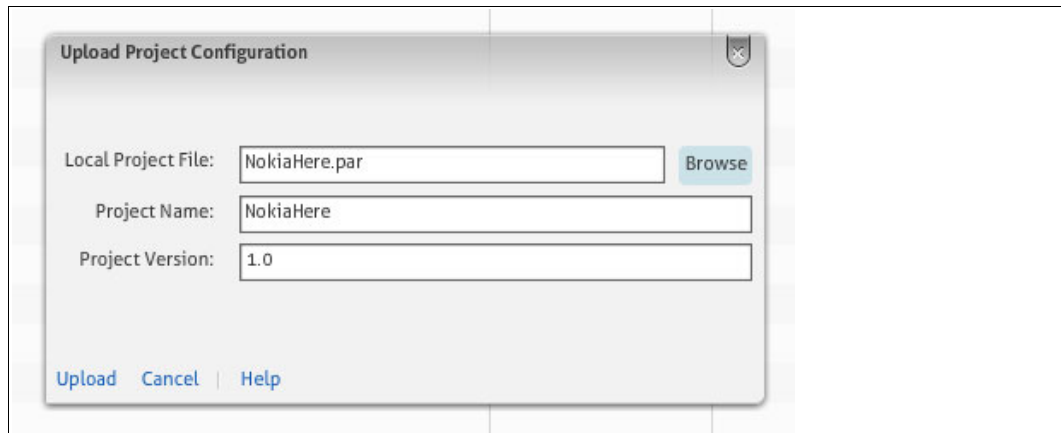


Figure 5-12 Uploading the project file

6. Click **Upload**. When the upload completes, the NokiaHere project is listed in the console as Undeployed, as shown in Figure 5-13.

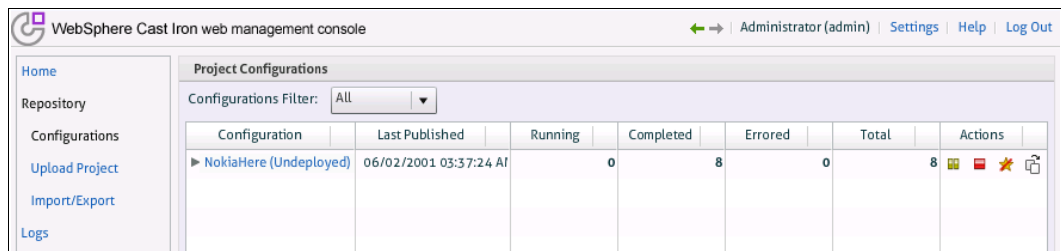


Figure 5-13 Uploaded configuration shown as undeployed

7. Switch to the detailed project view, shown in Figure 5-14, by clicking on the project name in the Configuration column.

Configuration Details

Summary

Configuration: NokiaHere

Orchestrations: 1

Status: Undeployed

Properties: 3

Last Published: 06/02/2001 09:22:10 AM

Assets: 0

Downtimes: 0

Download

Orchestrations (1)

Name	Status	Logging Level	Log Synchronously	Max Simultaneous Jobs
Orchestration	Enabled	Error Values	Disabled	10

Edit

Properties (3)

Name	Value
appIdIdentifier	ADD YOUR APP IDENTIFIER HERE
appToken	ADD YOUR APP TOKEN HERE
nokiaHereEnv	places.nlp.nokia.com

Edit

Figure 5-14 Cast Iron project details

8. Click **Edit** in the Properties section. In the Edit Configuration Properties dialog, enter your Nokia Here credentials in the Value column, using your app_id for the appIdIdentifier property and your app_code for the appToken property.
9. Click **Save** to return to the detailed project view.
10. Start the project by clicking the Play symbol that is located beside the Status field in the Summary section at the top of the screen. The status changes to Running after the project is deployed, as shown in Figure 5-15.

Configuration Details

Summary

Configuration: NokiaHere

Orchestrations: 1

Status: Running

Properties: 3

Last Published: 06/02/2001 09:22:10 AM

Assets: 0

Downtimes: 0

Download

Figure 5-15 Project displayed as running

11. Verify that the map service is working by invoking the following statement from a web browser, replacing `<cast_iron_data_ipaddress>` with the IP address of the data network of your virtual appliance and `<latitude>` and `<longitude>` with the latitude and longitude of the location for which you want to find nearby hotels:
`http://<cast_iron_data_ipaddress>/map?q=hotel&at=<latitude>,<longitude>`

For example, to search for hotels near Mainz, Germany, the URL is as follows:

`http://<cast_iron_data_ipaddress>/map?q=hotel&at=49.9945,8.2672`

With these steps completed, the WebSphere Cast Iron orchestration to retrieve the hotel information is deployed and tested, and can now be used by the application.

Configuring the IBM DataPower XG45 appliance

The IBM WebSphere DataPower Service Gateway XG45 appliance is used to provide security for MyLuggage. In contrast to LuggageTracker, the new application will be used by airline passengers outside of the corporate network. The XG45 appliance will function as a network security gateway, ensuring only authorized access from the mobile application to the internal corporate systems. Figure 5-16 illustrates the authentication flow.

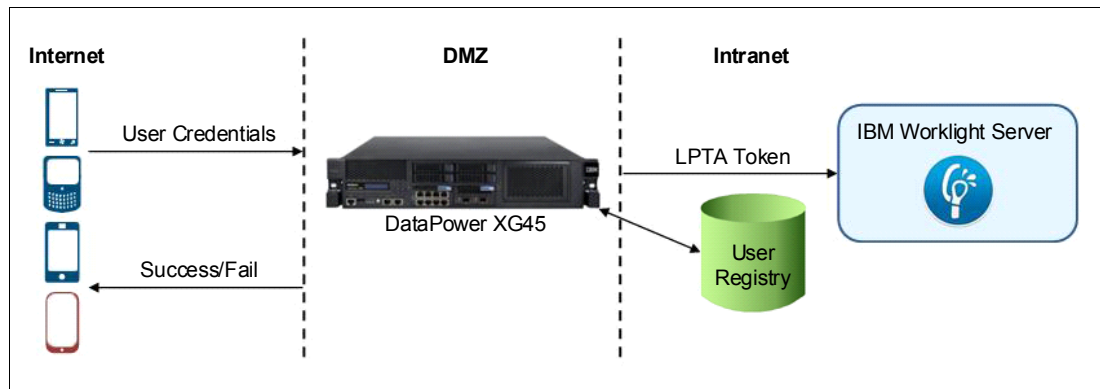


Figure 5-16 Authentication flow with the DataPower XG45 appliance

As a user logs in to the mobile application, the user credentials (user name and password) are sent to the XG45 appliance, which is located in the perimeter network (DMZ). The appliance authenticates incoming service requests against a user registry, which in this case contains airline customers (passengers) who have previously registered to use the company website. When a user is authenticated, a notice of a successful authentication is returned to the mobile application. If the user credentials are not valid, a notice of an authentication failure is returned to the mobile application.

Configuring an XML firewall to authenticate users

Airline Company A created an XML firewall policy on the XG45 appliance to implement its security model.

To configure the XML firewall to authenticate against the user registry, complete the following steps on the XG45 appliance:

1. Log in to the XG45 appliance using your administrator ID and password.
2. In the Control Panel, click **XML Firewall**.
3. To create a new XML firewall, click **Add Advanced**.
4. On the General tab of the Configure XML Firewall page, set the following attributes (shown in the red boxes in Figure 5-17 on page 225):
 - Firewall Name: PassengerFirewall
 - Comments: This is the firewall for registered passengers
 - Firewall Type: Static Backend (this creates a secure reverse proxy for Worklight Server)
 - Port Number: *<any unused port>* (7878 is shown in this example)

- Remote Host: <your_worklight_server_address> (9.111.27.228 is shown in this example)
- Remote Port: <your_worklight_server_port> (9080 is shown in this example)
- Response and Request Type: non-XML

Configure XML Firewall

General | Advanced | Stylesheet Params | Headers | Monitors | XML Threat Protection

Apply Cancel Help

General Configuration

Firewall Name
PassengerFirewall *

Comments
an example XML Firewall Service

Firewall Type
Static Backend *

XML Manager
default + ... *

Processing Policy
(none) + ... *

URL Rewrite Policy
(none) + ...

Back End

Remote Host
9.111.27.228 *

Remote Port
9080 *

Forward (Client) Crypto Profile
(none) + ...

Response Type
Non-XML

Front End

Local IP Address
0.0.0.0 Select Alias *

Port Number
7878 *

Reverse (Server) Crypto Profile
(none) + ...

Request Type
Non-XML

Figure 5-17 Configuring a new XML firewall on the XG45 appliance

5. Click the plus sign (+) beside the Processing Policy list to create a processing policy.
6. On the Configure XML Firewall Style Policy dialog, enter AuthenticateUser for the Policy Name and then click **New Rule**.
7. Change the rule name to authenticateUser and set the Rule Direction to **Client to Server**.

You notice that at first, the rule diagram contains a single Match action, as shown in Figure 5-18.

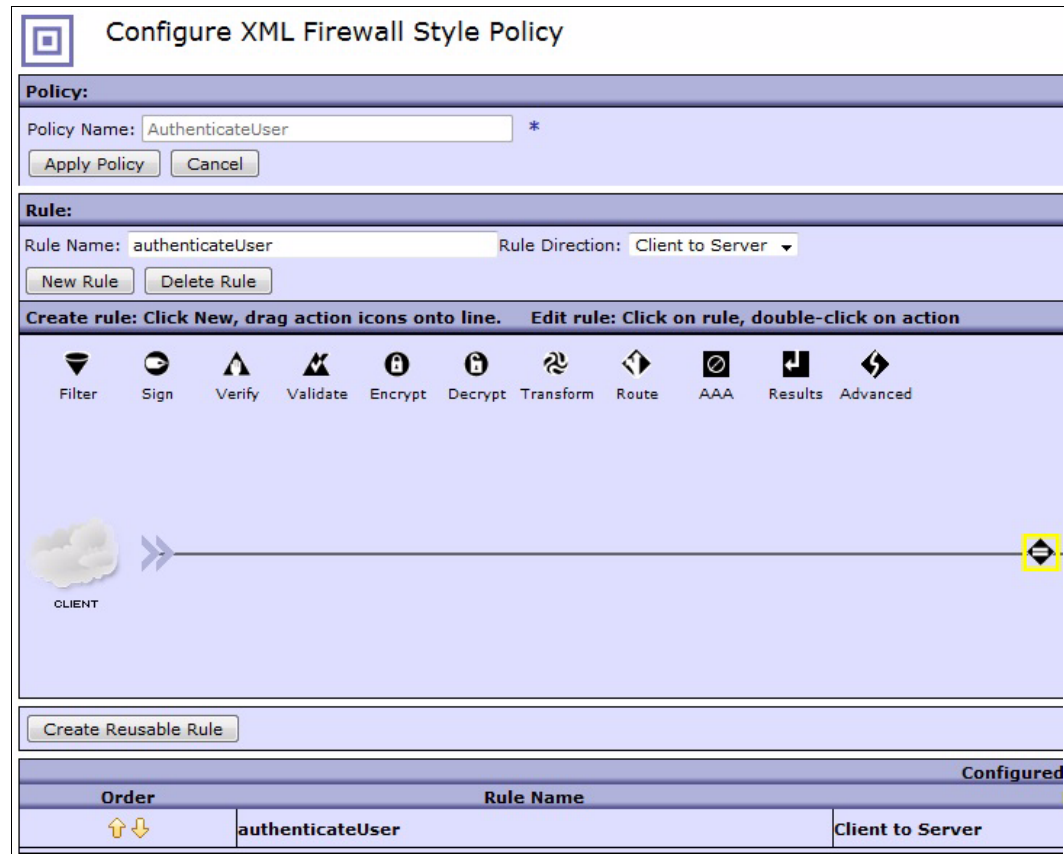


Figure 5-18 Creating the policy and first rule

8. Double-click the match rule icon (which is highlighted in yellow in the display) to display the Configure a Match Action dialog.
9. Click the plus sign (+) beside the Matching Rule property, as shown in Figure 5-19, to create a new matching rule.

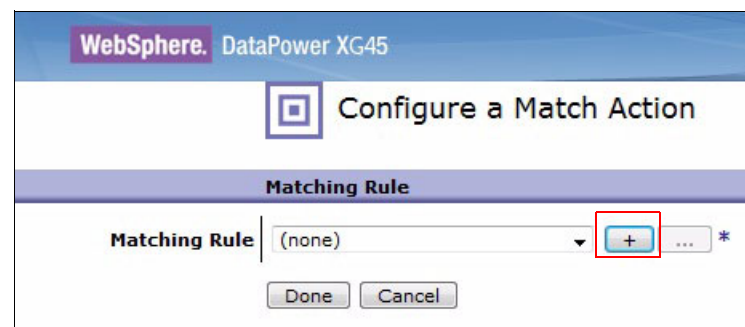


Figure 5-19 Creating the matching rule

10. The Configure Matching Rule dialog opens (Figure 5-20). Enter `authenticateUser_MatchingRule` for the name, and then click the **Matching Rule** link.

Configure Matching Rule

This configuration has been added and not yet saved.

Main **Matching Rule**

Matching Rule

Apply Cancel

Name *

Administrative State ☒ enabled ☐ disabled

Comments

Match with PCRE ☐ on ☒ off

Boolean Or Combinations ☐ on ☒ off

Figure 5-20 Configuring the matching rule

11. The Matching Rule table is added to the dialog. Click **Add** to configure the matching rule.
12. The Edit Matching Rule dialog opens (Figure 5-21). Select **URL** from the Matching Type list, enter `/authenticateUser` for the URL match, and click **Apply**.

WebSphere. DataPower XG45

Edit Matching Rule

Matching Type *

URL Match *

Apply Cancel

Figure 5-21 Editing the matching rule

13. The Configure Matching Rule dialog is redisplayed showing the URL matching rule that you just created in the table. Click **Apply**.
14. The Configure a Match Action dialog is redisplayed. Click **Done**.

With these steps completed, the Configure XML Firewall Style Policy dialog is redisplayed (Figure 5-22 on page 228).

The next part of the process involves adding an authorization, authentication, and auditing (AAA) action to the authenticateUser rule. To do this:

1. Select the **AAA** action icon. Drag it to the rule line, and place it to the right of the match action icon, as shown in Figure 5-22.

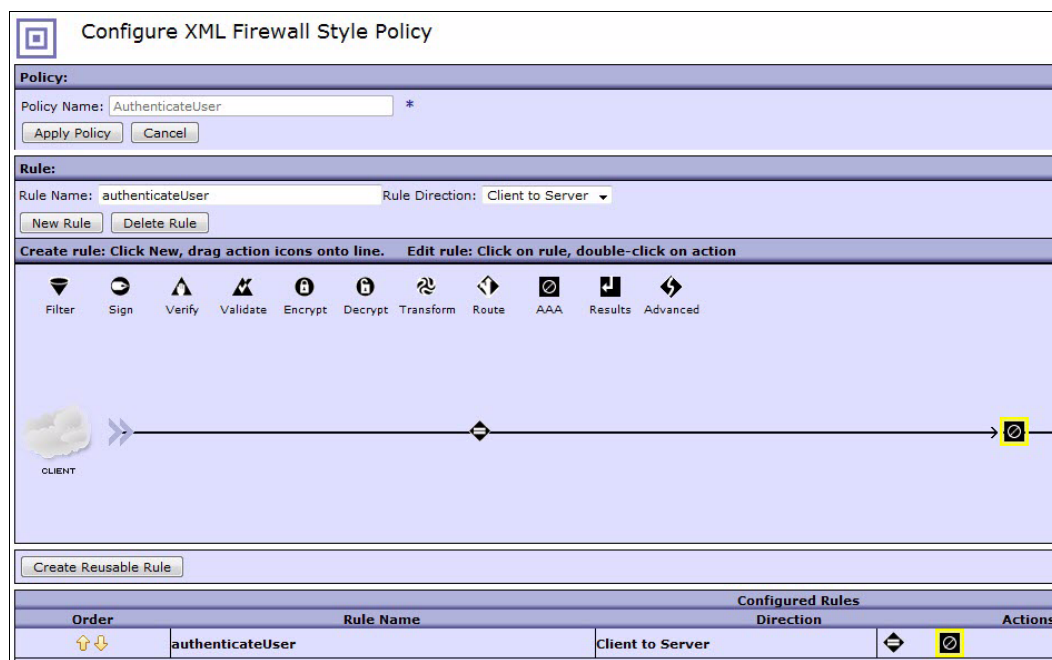


Figure 5-22 Adding the AAA action to the policy

2. Double-click the AAA action icon to open the Configure AAA Action dialog (Figure 5-23,).

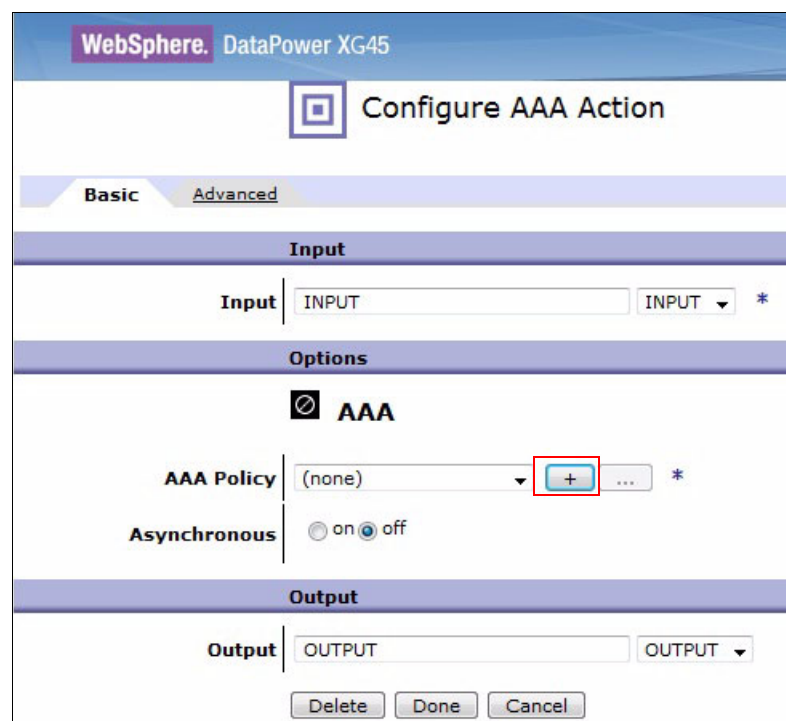


Figure 5-23 Configuring the AAA action

3. Select **INPUT** from the list beside the Input field (which displays INPUT in the text field) and select **OUTPUT** from the list beside the Output field (which displays OUTPUT in the text field).
4. Click the plus sign (+) beside the AAA Policy list to create a new AAA policy.
5. The Configure an Access Control Policy dialog opens (Figure 5-24). Enter `authenticate-user` for the AAA Policy Name and then click **Create**.
6. Select **HTTP Authentication Header** to extract the user's identity from the incoming request, and then click **Next**.

WebSphere DataPower XG45

Configure an Access Control Policy [Help](#)

AAA Policy Name: `authenticate-user`

Define how to extract a user's identity from an incoming request.

Identification Methods

- ☒ HTTP Authentication Header
- ☐ Password-carrying UsernameToken Element from WS-Security Header
- ☐ Derived-key UsernameToken Element from WS-Security Header
- ☐ BinarySecurityToken Element from WS-Security Header
- ☐ WS-SecureConversation Identifier
- ☐ WS-Trust Base or Supporting Token
- ☐ Kerberos AP-REQ from WS-Security Header
- ☐ Kerberos AP-REQ from SPNEGO Token
- ☐ Subject DN of the SSL Certificate from the Connection Peer
- ☐ Name from SAML Attribute Assertion
- ☐ Name from SAML Authentication Assertion
- ☐ SAML Artifact
- ☐ Client IP Address
- ☐ Subject DN from Certificate in the Message's signature
- ☐ Token Extracted from the Message
- ☐ Token Extracted as Cookie Value
- ☐ LTPA Token
- ☐ Processing Metadata
- ☐ Custom Template
- ☐ HTML Forms-based Authentication
- ☐ OAuth
- *

Figure 5-24 Configuring the policy to extract the user identity from the request header

7. The next step is to configure how to authenticate a specific user. The user registry in this scenario is an LDAP registry, so select the **Bind to Specified LDAP Server** method from the list, as shown in Figure 5-25, and then enter the directory service information for an LDAP connection in the fields at the bottom of the screen. If you use a different authentication method, select it from the list and enter the information as required.

Configure an Access Control Policy

AAA Policy Name: authenticate-user

Define how to authenticate the user.

Method

- ☐ Accept a SAML Assertion with a Valid Signature
- ☐ Accept an LTPA token
- ☒ Bind to Specified LDAP Server
- ☐ Contact a SAML Server for a SAML Authentication Statement
- ☐ Contact a WS-Trust Server for a WS-Trust Token
- ☐ Contact ClearTrust Server
- ☐ Contact Netegrity SiteMinder
- ☐ Contact NSS for SAF Authentication
- ☐ Contact Tivoli Access Manager
- ☐ Custom Template
- ☐ Pass Identity Token to the Authorize Step
- ☐ Retrieve SAML Assertions Corresponding to a SAML Browser Artifact
- ☐ Use an Established WS-SecureConversation Security Context
- ☐ Use certificate from BinarySecurityToken
- ☐ Use DataPower AAA Info File
- ☐ Use specified RADIUS Server
- ☐ Validate a Kerberos AP-REQ for the Correct Server Principal
- ☐ Validate the Signer Certificate for a Digitally Signed Message.
- ☐ Validate the SSL Certificate from the Connection Peer

*

LDAP Load Balancer Group (none) + ...

Host

Port 389

SSL Proxy Profile (none) + ...

LDAP Bind DN

LDAP Bind Password

Figure 5-25 Defining the LDAP connection to authenticate the users

8. Click **Next**.

- On the Configure an Access Control Policy page (Figure 5-26), select **URL sent by Client**. The completed post-processing options should look similar to what is shown in the figure.

Configure an Access Control Policy

AAA Policy Name: authenticate-user

Define how to extract the resources.

Resource Identification Methods

- ☐ URL Sent to Back End
- ☒ URL Sent by Client
- ☐ URI of Top level Element in the Message
- ☐ Local Name of Request Element
- ☐ HTTP Operation (GET/POST)
- ☐ XPath Expression
- ☐ Processing Metadata
- *

Define how to map resources.

Method | None *

Back Next Cancel

Figure 5-26 Defining the resource identification method

- Proceed through the wizard using the **Next** button to accept the default values.
- In the post-processing step of the wizard, ensure all the values are set to **off**.
- Click **Commit** and then **Done** to complete the configuration.
- Save the configuration changes.

The XML firewall is now configured to authenticate customers against a customer user registry and is ready to be tested.

Testing the XML Firewall configuration

The configuration now needs to be tested. Airline Company A tested the configuration using cURL, a command-line tool you can use to send or retrieve files using URL syntax. The cURL tool is available at no charge from the cURL website:

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

To invoke cURL for this scenario, use the following command, where `<username:password>` is a set of valid credentials that exist in the airline passenger user registry and `<dp_address:port>` is the appliance's IP address and the XML firewall port number (7878 in this example).

```
curl -v -u <username:password> http://<dp_address:port>:7878/authenticateUser
```

Figure 5-27 shows the result of the cURL test of the AAA policy to authenticate user login requests against the user registry.

```
$ curl -v -u username:password http://9.186.9.99:7878/authenticateUser
* About to connect() to 9.186.9.99 port 7878 (#0)
* Trying 9.186.9.99... connected
* Connected to 9.186.9.99 (9.186.9.99) port 7878 (#0)
* Server auth using Basic with user 'username'
> GET /authenticateUser HTTP/1.1
> Authorization: Basic a2VhcmFuQHphLmLibS5jb206QW15ZDFhbm51Mg==
> User-Agent: curl/7.19.7 (x86_64-redhat-linux-gnu) libcurl/7.19.7 NSS/3.13.1.0 zlib/1.2.3 libidn/1.18 libssh2/1.2.2
> Host: 9.186.9.99:7878
> Accept: */*
>
< HTTP/1.1 200 Good
< Authorization: Basic a2VhcmFuQHphLmLibS5jb206QW15ZDFhbm51Mg==
< User-Agent: curl/7.19.7 (x86_64-redhat-linux-gnu) libcurl/7.19.7 NSS/3.13.1.0 zlib/1.2.3 libidn/1.18 libssh2/1.2.2
< Host: 9.186.9.99:7878
< Via: 1.1 test
< X-Client-IP: 9.145.196.7
< Content-Type: text/xml
< Transfer-Encoding: chunked
<
* Connection #0 to host 9.186.9.99 left intact
* Closing connection #0
```

Figure 5-27 Testing of the AAA policy to authenticate the user using cURL

The XML firewall is now tested and is ready to authenticate users for MyLuggage.

5.3 Changes to the back-end services

Two changes are required to the back-end services to support the new functions in MyLuggage.

The new application will use the airline’s existing back-end services to obtain the current status of a passenger’s bag and record the passenger’s preferred delivery address. But when that delivery address will be a hotel, the airline wants to allow passengers to locate their hotel on a map instead of having to enter the specific address using the keyboard. Offering this feature requires that hotel information be obtained from a third-party mapping service. Obtaining the hotel information will be done using a new adapter.

The second change is to the existing authentication adapter. Although the authentication of user credentials for MyLuggage will be done using the DataPower appliance, Worklight Server still needs to get the user identity set after authentication is completed. This can be accomplished by adding another procedure to the existing adapter.

5.3.1 Creating the hotel adapter

The third-party mapping service will use IBM WebSphere Cast Iron as an integration broker, which is described in 5.3, “Changes to the back-end services” on page 232.

The hotel information that is obtained from the third-party service will be cached. There are several reasons to use caching:

- ▶ Invoking third-party services takes time, which can slow the application.
- ▶ Third-party services may charge on a per-request basis, which can add to the overall cost of the new solution.
- ▶ Only a limited number of airport locations are likely to be searched, and passengers will search for the same hotels repeatedly.

Costs can be reduced and response times improved by determining if specific hotel information is already cached before invoking the third-party service.

To satisfy these requirements, the server component developer must create a new IBM Worklight adapter, which will use IBM WebSphere Cast Iron to invoke the third-party service to obtain the hotel information. Caching capabilities are then added to cache the hotel information.

There are three stages to implementing the hotel information adapter:

- ▶ Creating the adapter
- ▶ Adding caching capabilities
- ▶ Testing the adapter

Creating the adapter

The hotel information Cast Iron adapter is part of the `AirlineAdapterProject` that also contains the adapters created in the previous chapter. To create the hotel information adapter, use the following steps:

1. Start IBM Worklight Studio if it is not already started.
2. Expand the `AirlineAdapterProject` project.
3. Right-click the `adapters` folder and create a new adapter by selecting **File** → **New** → **Worklight Adapter**.
4. In the New Worklight Adapter dialog, select **Cast Iron Adapter** from the Adapter type list, and enter `MapAdapter` for the Adapter name.
5. Click **Finish** to create the adapter. A new folder named `MapAdapter` is created and within it are several auto-generated files similar to the files that were generated when the previous adapters were created. You can delete the `ci-filtered.xml` file because it is not used in this scenario.
6. The `MapAdapter.xml` file is automatically opened in the adapter editor. If this does not occur, open the file by right-clicking on the file and selecting **Open With** → **Adapter Editor**.
7. Switch to the Source tab if it is not shown.
8. In the XML source, find the `<wl:adapter>` tag block that was created when the file was auto-generated and replace it with the XML shown in Example 5-1 on page 234. When doing this, replace `server_name` and `server_port` with the server URL and port of your WebSphere Cast Iron server. This XML defines the method, `getHotels`, that will be called when the adapter is invoked.

Example 5-1 Replacement adapter tag block in MapAdapter.xml

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

  <displayName>MapAdapter</displayName>
  <description>MapAdapter</description>
  <connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>http</protocol>
      <domain>server_name</domain>
      <port>server_port</port>
    </connectionPolicy>
    <loadConstraints maxConcurrentConnectionsPerNode="2" />
  </connectivity>

  <procedure name="getHotels"/>

</wl:adapter>
```

9. Save and close the MapAdapter.xml file.

The new function, getHotels, is implemented in the MapAdapter-impl.js file and is called from the adapter definition (see Example 5-1). To add this function, use the following steps:

1. Open the MapAdapter-impl.js file in Rich Page Editor.
2. Replace the auto-generated functions with the JavaScript shown in Example 5-2. The getHotels function implements the procedure defined in the adapter XML shown in the previous example.

The input parameters of the getHotels() function are the latitude and longitude coordinates to be used in the search for nearby hotels. The JavaScript creates the input variable that is passed to Worklight Server and used to invoke the Cast Iron service. The Cast Iron service is invoked using an HTTP GET request to the following URL:

`http://<server>:<port>/map?q=hotel&at=<latitude>,<longitude>`

Example 5-2 Adding the getHotels function

```
function getHotels(latitude, longitude){

  var input = {
    method : 'get',
    appName : '/',
    path : 'map?q=hotel&at='+latitude+','+longitude,
    returnedContentType : 'json'
  };

  return WL.Server.invokeCastIron(input);
}
```

3. Save and close the MapAdapter-impl.js file.

The adapter is now able to call the third-party server to obtain hotel information. You can test it now, or wait until after the caching is added.

Adding caching capabilities

The adapter will integrate with WebSphere eXtreme Scale as a caching server to improve application response times. This caching approach assumes that hotel location information is generally static and that multiple customers will be searching for the same hotels.

Defining the cache key is critical to gaining the full benefit of caching. Exact longitude and latitude coordinates, if used as a key, would be too precise and therefore generate too few cache hits. So the user's coordinates are rounded to two decimal places, which is unlikely to affect the customer but will result in much-improved hit rates and response times.

The tasks required to add caching to the MapAdapter are as follows:

- ▶ Creating the client-side caching classes
- ▶ Adding a new procedure in the adapter
- ▶ Creating the JavaScript function that calls the client-side caching classes

Creating the client-side caching classes

Airline Company A created the following three Java classes to handle the connection to the WebSphere eXtreme Scale server, data retrieval, and updating of the cached data:

- ▶ WLCache: converts the cache objects to and from JSON
- ▶ ConnectXS: acts as a client proxy for connecting to the caching server
- ▶ ExternalXS: an abstraction layer for the caching operations

The Java source for these files is in “WebSphere eXtreme Scale caching integration files” on page 356. Create these three Java files, compile them, and copy the class files into the server/java folder of the AirlineAdapterProject project. During compilation, you must add the `osgi.jar` and `js.jar` files (from the server/lib folder of the AirlineAdapterProject project) to your build path.

For more information about using WebSphere eXtreme Scale as a caching solution for IBM Worklight applications, see the *eXtreme Scale caching with Worklight 5.0* developerWorks article:

https://www.ibm.com/developerworks/mydeveloperworks/blogs/worklight/entry/combining_worklight_and_extreme_scale

Updating the adapter definition

To enable caching at the adapter level, add another procedure call to obtain the cached hotel information, use the following steps:

1. Open the `MapAdapter.xml` file in Rich Page Editor.
2. Go to the Source tab.

3. Add a new procedure called `getCachedHotels` beneath the `getHotels` procedure, as shown in bold in Example 5-3.

Example 5-3 Adding the `getCachedHotels` procedure to the adapter definition

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

  <displayName>MapAdapter</displayName>
  <description>MapAdapter</description>
  <connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>http</protocol>
      <domain>server_name</domain>
      <port>server_port</port>
    </connectionPolicy>
    <loadConstraints maxConcurrentConnectionsPerNode="2" />
  </connectivity>

  <procedure name="getHotels"/>
  <procedure name="getCachedHotels"/>

</wl:adapter>
```

4. Save and close the `MapAdapter.xml` file.

Creating the JavaScript function to cache the data

The new function, `getCachedHotels`, is implemented in the `MapAdapter-impl.js` file and is called from the adapter definition (see Example 5-3).

The function creates the cache key from the latitude and longitude coordinates passed in as parameters to the function call, and then uses the three client-side caching classes created in “Creating the client-side caching classes” on page 235 to add data to the cache and retrieve cached data when requested.

To create the new `getCachedHotels` function, use the following steps:

1. Open the `MapAdapter-impl.js` file in Rich Page Editor.
2. Add the `getCachedHotels` function, shown in Example 5-4 on page 237, beneath the `getHotels` function. The `getCachedHotels` function implements the new procedure defined in the adapter XML shown in the previous example and adds caching to the previous `getHotels` function (which is now not used but was not removed in this scenario).

The cache key is created first and contains the latitude and longitude coordinates rounded to two decimal places. When a new request is received, the `WLCache` class connects to the caching grid that was created in 5.2.2, “Changes to the pre-production environment” on page 220 to check for cached hotel information based on the rounded coordinates in the cache key. If the cache key exists, the cached value (the map of hotels located near the submitted coordinates) is returned in JSON format. If the cache key is not found, the Cast Iron adapter is called to get the necessary hotel information from the third-party source. The information that is retrieved is added to the cache and then returned to the application.

Example 5-4 Adding the caching function

```
function getCacheHotels(latitude, longitude) {

    // Create the caching key
    var inputKey = latitude.toFixed(2)+'/'+longitude.toFixed(2);

    // Connect to eXtreme Scale
    var wLCache =
com.ibm.worklight.example.extremescale.WLCache("caching.server", 2809, "Grid",
"Map1");

    if (wLCache.getString(inputKey)) {      //object exists, return cached value
        return JSON.parse(wLCache.getString(inputKey));
    }

    else {    // invoke Cast Iron to retrieve hotel information
        var input = {
            method : 'get',
            appName : '/',
            path : 'map?q=hotel&at='+inputKey,
            returnedContentType : 'json'
        };

        var result = WL.Server.invokeCastIron(input);
        if(result != null){      // adding new value to the cache
            wLCache.putString(inputKey, JSON.stringify(result));
        }
    }
    return result;
}
```

3. Save and close the MapAdapter-impl.js file.

The adapter is now updated to use WebSphere eXtreme Scale to cache the hotel information and can be tested.

Testing the adapter

The adapter is deployed and tested using the same procedures described in “Testing the adapter” on page 103, and using various longitude and latitude combinations.

5.3.2 Updating the authentication adapter

Even though mobile application authentication is handled by the DataPower XG45 appliance, the active user in IBM Worklight must still be set to the identity of the authenticated user. To minimize development and maintenance costs, the authentication adapter created in 4.7.2, “Creating the authentication adapter” on page 91, was updated to add a new procedure to set the active user instead of creating a new adapter.

There are three stages to updating the authentication adapter:

- ▶ Updating the adapter definition
- ▶ Creating the JavaScript function
- ▶ Testing the adapter

Updating the adapter definition

The existing authentication adapter, named `LuggageTrackerAdapter`, must be updated to add a new procedure, which will be invoked by the client-side authentication process after the actual user credential validation has been completed. This new procedure is used to set the active user in Worklight Server.

To add the new procedure, use the following steps:

1. Start IBM Worklight Studio if it is not already started.
2. Expand the `AirlineAdapterProject` project and then expand the adapters and `LuggageTrackerAdapter` folders.
3. Open the `LuggageTrackerAdapter.xml` file in Rich Page Editor.
4. Go to the Source tab.
5. Add a new procedure named `setUserIdentity` beneath the `submitAuthentication` procedure, as shown in bold in Example 5-5.

Example 5-5 Adding the `setUserIdentity` procedure to the `LuggageTrackerAdapter` definition

```
<wl:adapter name="LuggageTrackerAdapter"
  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>LuggageTrackerAdapter</displayName>
  <description>LuggageTrackerAdapter</description>
  <connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>http</protocol>
      <domain>localhost</domain>
      <port>8088</port>
    </connectionPolicy>
    <loadConstraints maxConcurrentConnectionsPerNode="2" />
  </connectivity>
  <procedure audit="true" name="submitAuthentication"/>
  <procedure audit="true" name="setUserIdentity"/>
  <procedure audit="true" name="onLogout"/>
</wl:adapter>
```

6. Save and close the `LuggageTrackerAdapter.xml` file.

Creating the JavaScript function

The new function, `setUserIdentity`, is implemented in the `LuggageTrackerAdapter-impl.js` file and is called from the adapter definition (see Example 5-5).

Follow these steps:

1. Open the `LuggageTrackerAdapter-impl.js` file in Rich Page Editor.
2. Add the JavaScript for the `setUserIdentity` function, shown in Example 5-6 on page 239, to the file beneath the `submitAuthentication` function.

Example 5-6 JavaScript for the setUserIdentity function

```
function setUserIdentity(username, password){
    if (username){
        WL.Logger.debug("setUserIdentity");
        var userIdentity = {
            userId: username,
            displayName: username,
            attributes: {
                userType: "Passenger"
            }
        };

        WL.Server.setActiveUser("LuggageTrackerRealm", userIdentity);

        return {
            isLoginSuccessful:true,
            authRequired: false
        };
    }
    WL.Server.setActiveUser("LuggageTrackerRealm", null);
    return onAuthRequired(null, "Invalid login credentials");
}
```

3. Save and close the LuggageTrackerAdapter-impl.js file.

The adapter is now updated to set the active user within Worklight Server.

Testing the adapter

The adapter is deployed and tested using the same procedures described in “Testing the adapter” on page 103, using both valid and invalid credentials.

5.4 Creating the new mobile application

The new mobile application, MyLuggage, will have a completely new user interface created using Dojo Mobile instead of jQuery. The use of Dojo Mobile is intended only to show how to use it; there is no technical advantage to using one product instead of the other for development of this specific application.

The new customer-facing application will also incorporate a new security mechanism using the DataPower XG45 appliance (configured in “Configuring the IBM DataPower XG45 appliance” on page 224), and add a native map component to assist users in providing their delivery address.

5.4.1 Creating the user interface using Dojo

The MyLuggage application consists of four screens, each of which contains specific elements. These elements are almost identical to those in LuggageTracker, although minor differences exist. The screens and elements in the MyLuggage application are as follows:

- ▶ Login
 - Text input field for User Name
 - Text input field for Password
 - Login button
- ▶ Luggage ID input
 - Text input field for Luggage ID
 - Scan Bar Code button
 - Retrieve Information button
 - Logout button
- ▶ Luggage information
 - Text input field for Current Status
 - Text input field for Owner Name
 - Text input field for Current Location
 - Text input field for Next Destination
 - Text input field for Final Destination
 - Text input field for Last Update
 - Text area field for Comments
 - Change Delivery Address button
 - Close button
- ▶ Luggage delivery address
 - Text input field for Luggage ID
 - Text input field for Name
 - Text input field for Address Line 1
 - Map button
 - Text input field for Address Line 2
 - Text input field for City
 - Text input field for State
 - Text input field for Zip Code
 - Text input field for Phone Number
 - Text area field for Comments
 - Save and return button
 - Cancel button

For purposes of this book, detailed instructions are provided only for creating the Login and Luggage ID input screens using Dojo Mobile. Abbreviated instructions for creating the Luggage information and Luggage delivery address screens using Dojo Mobile are supplied at the conclusion of this section.

Creating the new hybrid application

The user interface for MyLuggage is created using Dojo Mobile in the same IBM Worklight Studio project (CompanyAirlines) that is used for the original LuggageTracker application. During the creation of the project (see “Creating the user interface project” on page 106), the hybrid application for LuggageTracker was created. Because MyLuggage will use the same project, a new hybrid application must be created for the new application.

Use the following steps:

1. Start IBM Worklight Studio if it is not already started.
2. Expand the CompanyAirlines project.
3. Right-click the apps folder and select **New** → **Worklight Hybrid Application**, as shown in Figure 5-28.

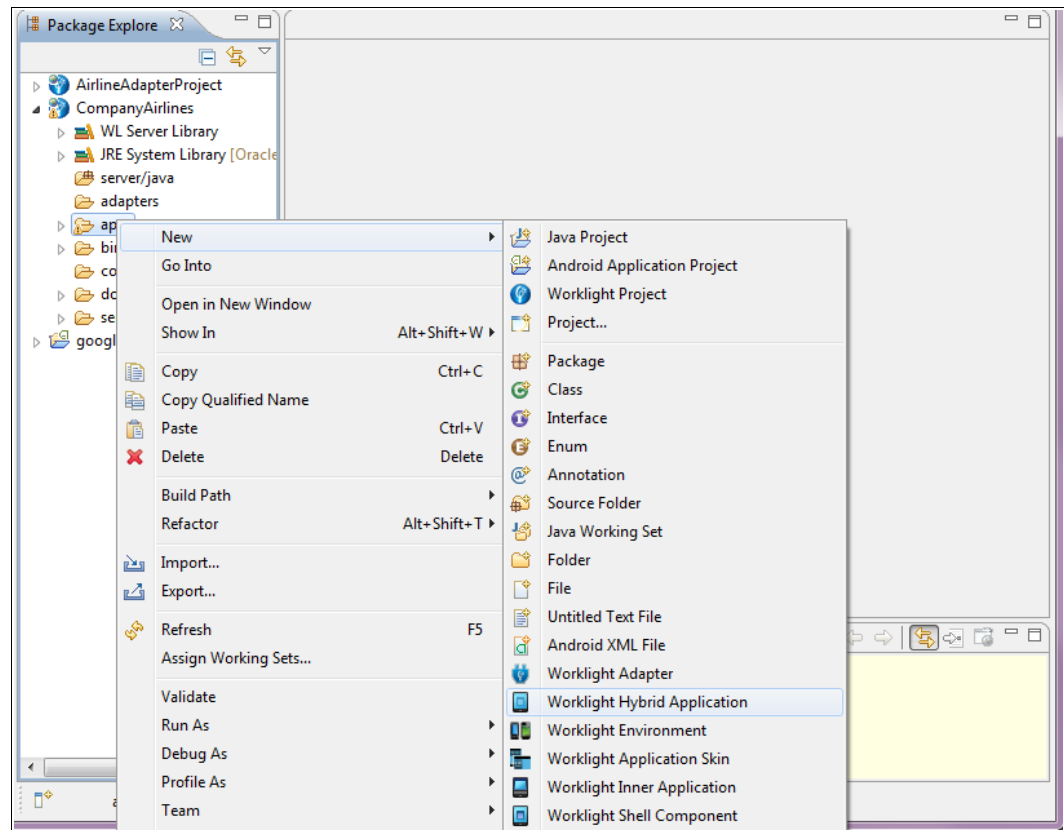


Figure 5-28 Creating a new Worklight Hybrid Application

4. In the dialog that is displayed, enter LuggageTrackerDojo in the Application name field and select the **Add Dojo Toolkit** check box, as shown in Figure 5-29.

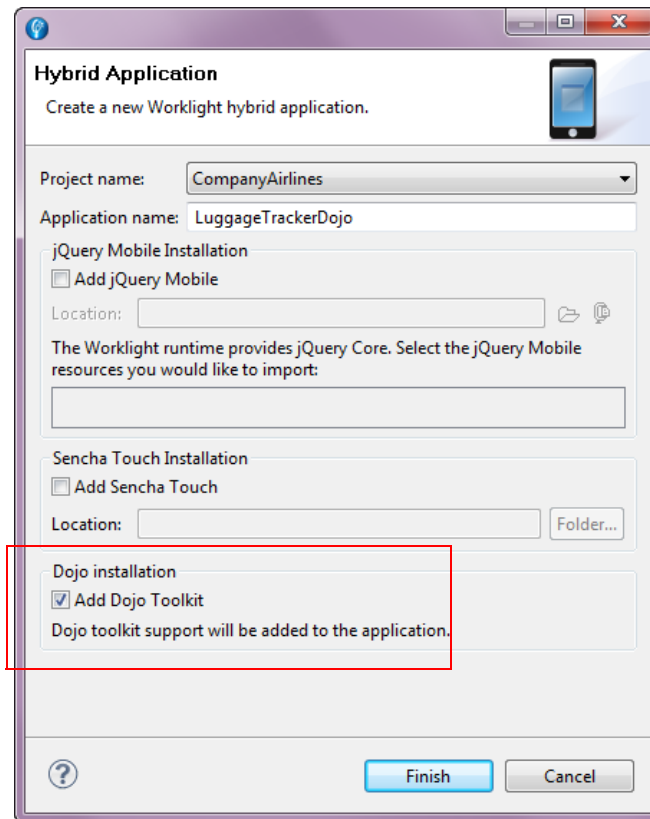


Figure 5-29 Specifying the application name and adding the Dojo Toolkit

5. Click **Finish**. The CompanyAirlines project in Project Explorer now shows both applications, LuggageTracker and LuggageTrackerDojo (the MyLuggage application), as shown in Figure 5-30.

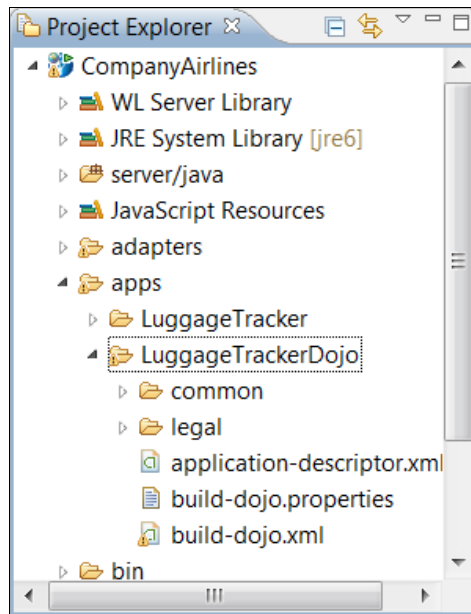


Figure 5-30 New hybrid application displayed in Project Explorer

The project name used within Worklight Studio is LuggageTrackerDojo but the display name should be MyLuggage. To change the display name of the application, use the following steps:

1. If it is not already open, double-click the `application-descriptor.xml` file in Project Explorer to open it in Application Descriptor Editor.
2. On the Design tab, change the Display name attribute from LuggageTrackerDojo to MyLuggage. Leave the ID attribute unchanged.
3. You may also change the Description by editing the Description attribute.
4. Save and close the `application-descriptor.xml` file.

Building the user interface screens

To create the user interface screens, you edit the mobile application main file, `LuggageTrackerDojo.html`, which was automatically generated when the LuggageTrackerDojo application was created. To open this file and prepare your Worklight Studio environment for creation of the various screens, use the following steps:

1. In Project Explorer, expand the CompanyAirlines project until the files in the LuggageTrackerDojo folder are visible.
2. Open the `LuggageTrackerDojo.html` file, which displays in the Rich Page Editor portion of the Design perspective.
3. Ensure that the Palette view is open, and if not, open it by selecting **Window** → **Open view** → **Other** and then selecting **Palette** from the dialog. The Dojo Mobile Widgets section is included in the Palette because the Dojo Toolkit was added to the application when it was created.

Your Worklight Studio environment is now ready for you to proceed with creating the user interface screens. Each application screen is defined within a series of screen-specific `ScrollableView` widgets that are placed between the header and footer in Rich Page Editor.

The application hides and shows these ScrollableView widgets to display particular screens to the user.

Tip: As you make changes in Rich Page Editor for the Dojo-based user interface, if the changes are not being reflected correctly in the Rich Page Editor, click **refresh** on the toolbar to refresh the display.

Common header and footer

All four screens of the mobile application are defined in the `LuggageTrackerDojo.html` file, and all have the same header and footer. The header contains the name of the application, My Luggage; the footer is a solid black bar that is shown only for cosmetic purposes.

Both the header and footer are defined with Dojo Mobile Heading widgets in Rich Page Editor. To add the header and footer, use the following steps:

1. Find the Heading widget in the Dojo Mobile Widgets section of the Palette, drag it to the Design section, and then drop it where the hover help text says **Insert into <body> 'content' above ScrollableView 'view0'**, as shown in Figure 5-31. This placement ensures that the header sits outside of the scrollable area of the application.

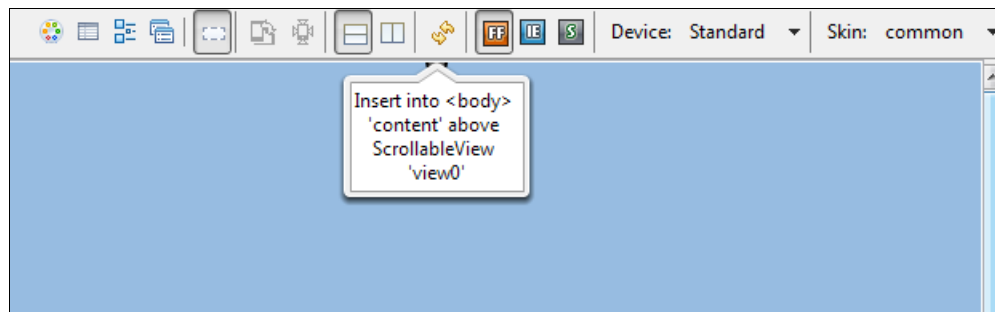


Figure 5-31 Dropping the Dojo Mobile Heading widget into Rich Page Editor

2. Change the header text from Heading to My Luggage, either by editing the source code in the Source section or by double-clicking on the heading widget, changing the text, and then clicking a space outside of the widget.
3. Drag another Heading widget into the Design section and drop it below the ScrollableView, using the hover help text to guide the placement of the widget.
4. Remove the default text from the second Heading widget using one of the methods described in Step 2, so the footer shows no text.
5. To ensure that the footer is fixed at the bottom of the device screen, in the HTML source code, add the text `data-dojoprops="fixed: 'bottom'"` before the `</h1>` end-tag for the footer. When complete, the body section of the HTML source looks like Example 5-7 on page 245.

Example 5-7 Body section of HTML source after adding the common header and footer

```
<body id="content" style="display: none;">
  <h1 data-dojo-type="dojox.mobile.Heading"
    data-dojo-props="label:'My Luggage'" ></h1>
  <div data-dojo-type="dojox.mobile.ScrollableView" id="view0"
    data-dojo-props="selected:true"></div>
  <h1 data-dojo-type="dojox.mobile.Heading" data-dojo-props="fixed:
'bottom'"></h1>

  LuggageTrackerDojo
  <!--application UI goes here-->
  <script src="js/initOptions.js"></script>
  <script src="js/LuggageTrackerDojo.js"></script>
  <script src="js/messages.js"></script>
</body>
```

6. Locate the line above the comment line that reads `<!-- application UI goes here...>` and you see the hybrid application name `LuggageTrackerDojo` was automatically added to the HTML. Remove the text because it is not needed in the display.

With the header and footer now created, you are ready to lay out each individual screen.

Login screen

The Login screen consists of two `TextBox` widgets and a `Button` widget, all contained within the `ScrollableView` widget that was auto-generated when the hybrid application was created. To create the Login screen, use the following steps:

1. Change the ID property of the auto-generated `ScrollableView` widget from `view0` to `loginView`. This can be done on the Tag tab of the Properties view, or by editing the HTML source code in Rich Page Editor.
2. Drag a `RoundRect` widget from the Palette and drop it into the `ScrollableView` widget. The `RoundRect` widget makes it easier to size and align widgets for optimal viewing.
3. On the Source tab, locate the HTML source code for the `RoundRect` widget (look for `<div data-dojo-type="dojox.mobile.RoundRect"></div>`) and add the following HTML table before the `</div>` tag:

```
<table style="width:100%">
  <tr><td>a</td></tr>
  <tr><td>b</td></tr>
  <tr><td>c</td></tr>
</table>
```

The `TextBox` and `Button` widgets will be placed within this table. The letters a, b, and c are temporary placeholders to make the next steps easier.

4. Refresh the Rich Page Editor by clicking the **Refresh** icon in the toolbar (highlighted by the red box in Figure 5-32). The screen is similar to what is shown in the figure.

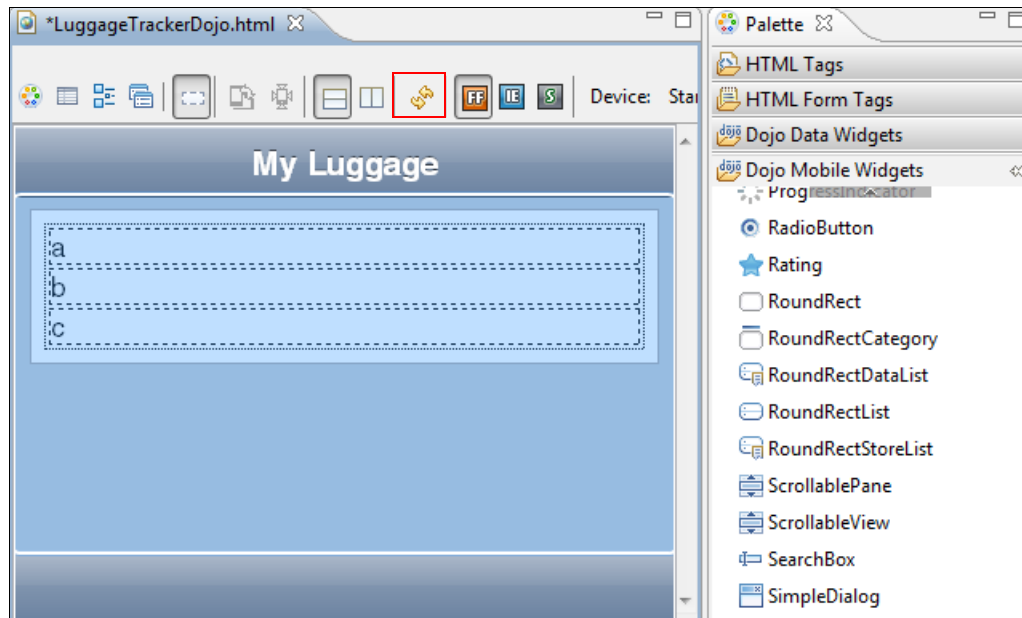


Figure 5-32 Login screen (in progress) with header, footer, and table for text and button widgets

5. Add the field for the User Name:
 - a. Drag a TextBox widget from the Palette and drop it into the first row of the table (within the <td> tag). The hover text similar to that shown in Figure 5-31 on page 244 can help you drop the widget in the correct place. Remove the placeholder letter from the row.
 - b. On the Design tab, click the text field that you just dropped into the first row of the table in step a (this selects it) and then switch to the Tag tab of the Properties view to display the tag properties of the selected text field. Change the ID property of the TextBox widget to username. If you use a different ID for the TextBox widget, you will have to make the same change in the JavaScript functions that you will add later.
 - c. To make the User Name field expand to fill the available horizontal space on the device screen, select the Styles tab of the Properties view and enter width:100% into the Properties attribute.
 - d. With Dojo Mobile, the TextBox widget has no label and there is no Label widget. So to properly label the input field users will see in the application, you must manually add a label to the HTML source, so that the associated HTML table row has the following text:


```
<tr><td><label for="username">User Name:</label><input
data-dojo-type="dojox.mobile.TextBox" id="username"
style="width:100%"></td></tr>
```
 - e. Click **Refresh** to refresh the Rich Page Editor preview, which looks like Figure 5-33 on page 247.

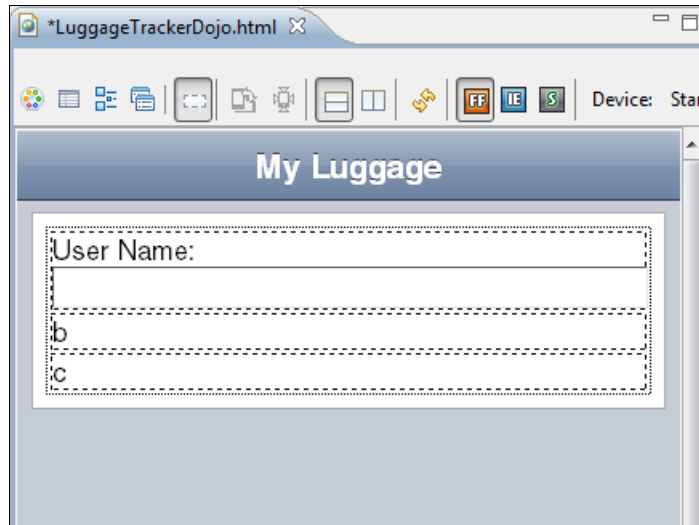


Figure 5-33 Login screen (in progress) after addition of TextBox widget for User Name

6. Add the field for the Password:
 - a. Repeat the process outlined in step 5 on page 246, adding a TextBox widget to the second row of the table and changing the widget's label text to Password and the id to password.
 - b. On the Tag tab of the Properties view, set the Input type to Password so that the password is displayed as it is typed.
7. Add the Login button:
 - a. Drag a Button widget from the Palette and insert it into the <td> tag for the last row of the table.
 - b. On the Tag tab of the Properties view, change the ID property to loginBtn.
 - c. On the Styles tab of the Properties view, enter width:100% for the Properties attribute.
 - d. Change the text for the button from Label to Login and remove the placeholder letter from the row, either in Rich Page Editor or by editing the HTML source.

The Design view in Rich Page Editor now shows the completed Login screen (Figure 5-34). The body section of the HTML source looks like what is shown in Example 5-8 on page 248.

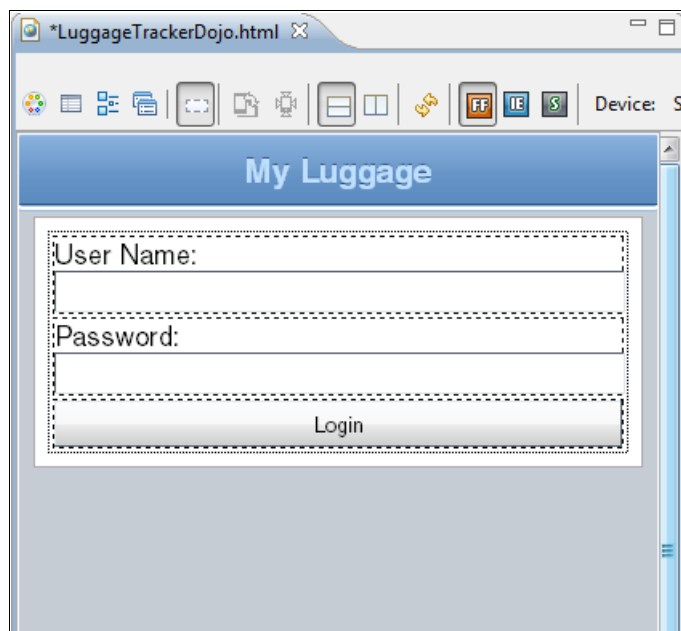


Figure 5-34 Completed Login screen displayed in Rich Page Editor

Example 5-8 Body section of HTML source after adding code for Login screen

```
<body id="content" style="display: none;">
  <h1 data-dojo-type="dojox.mobile.Heading"
    data-dojo-props="label:'My Luggage'" ></h1>
  <div data-dojo-type="dojox.mobile.ScrollableView" id="loginView"
    data-dojo-props="selected:true">
    <div data-dojo-type="dojox.mobile.RoundRect">
      <table style="width:100%">
        <tr><td><label for="username">User Name:</label>
          <input data-dojo-type="dojox.mobile.TextBox" id="username"
style="width:100%">
        </td></tr>
        <tr><td><label for="password">Password:</label>
          <input data-dojo-type="dojox.mobile.TextBox" id="password"
style="width:100%" type="password">
        </td></tr>
        <tr><td><button data-dojo-type="dojox.mobile.Button" style="width:100%"
id="loginBtn">Login</button>
        </td></tr>
      </table></div>
    </div>
  <h1 data-dojo-type="dojox.mobile.Heading" data-dojo-props="fixed: 'bottom'"></h1>

  LuggageTrackerDojo
  <!--application UI goes here-->
  <script src="js/initOptions.js"></script>
  <script src="js/LuggageTrackerDojo.js"></script>
  <script src="js/messages.js"></script>
</body>
```

Using the Mobile Navigation view

Unlike when using JQuery, when using Dojo Mobile, Rich Page Editor shows only one screen at a time. To change the screen that is displayed, or to create new screens, open the Mobile Navigation view using these steps:

1. Select **Window** → **Show View** → **Other** from the Worklight Studio menu bar.
2. Expand the Web folder in the Show View dialog.
3. Select **Mobile Navigation** and click **OK**.

The Mobile Navigator view is displayed and shows any screens that are already created (in this case, the Login screen, as shown in Figure 5-35).

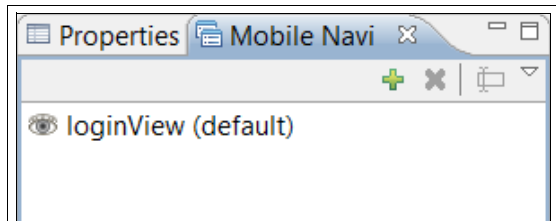


Figure 5-35 The Mobile Navigation view showing screens that have been created

Luggage ID input screen

The Luggage ID input screen consists of one TextBox widget and three Button widgets, all contained within a ScrollableView widget. When the application was created, contained a single screen (defined within a ScrollableView widget) that we renamed in step 1 on page 245. To create a new ScrollableView widget for the Luggage ID input screen, use the following steps:

1. Add a ScrollableView widget for the Luggage ID input screen by clicking the plus sign (+) on the Mobile Navigation view toolbar. This displays the Dojo Mobile View wizard.
2. In the Add Dojo Mobile View wizard, select **ScrollableView**, as shown in Figure 5-36 on page 250, and then click **Next**.

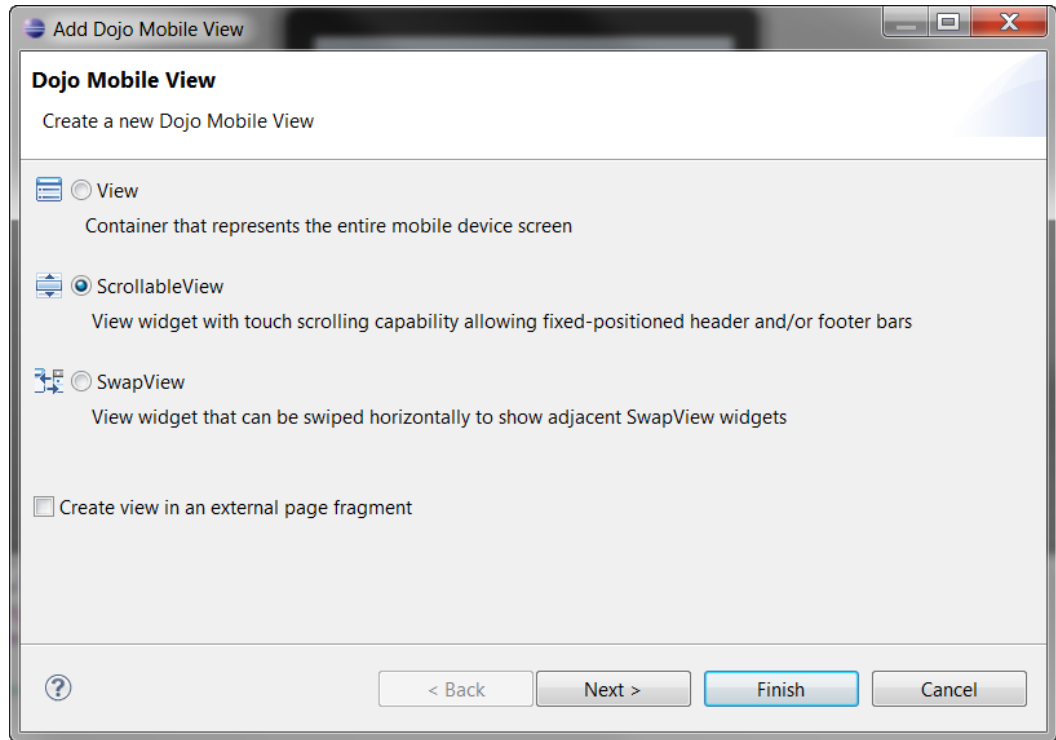


Figure 5-36 Selecting ScrollableView from the Add Dojo Mobile View wizard

3. On the next page of the wizard, change the ID attribute to `luggageInputView` and click **Next**.
4. On the next page of the wizard, specify where the ScrollableView is to be placed:
 - a. Click **Select** beside the Location field.
 - b. In the Location dialog, select **Next Sibling** from the Relation drop-down list and then choose **ScrollableView “loginView”** from the list of target elements, as shown in Figure 5-37 on page 251. These actions place the new scrollable view immediately beneath the ScrollableView for the Login screen.

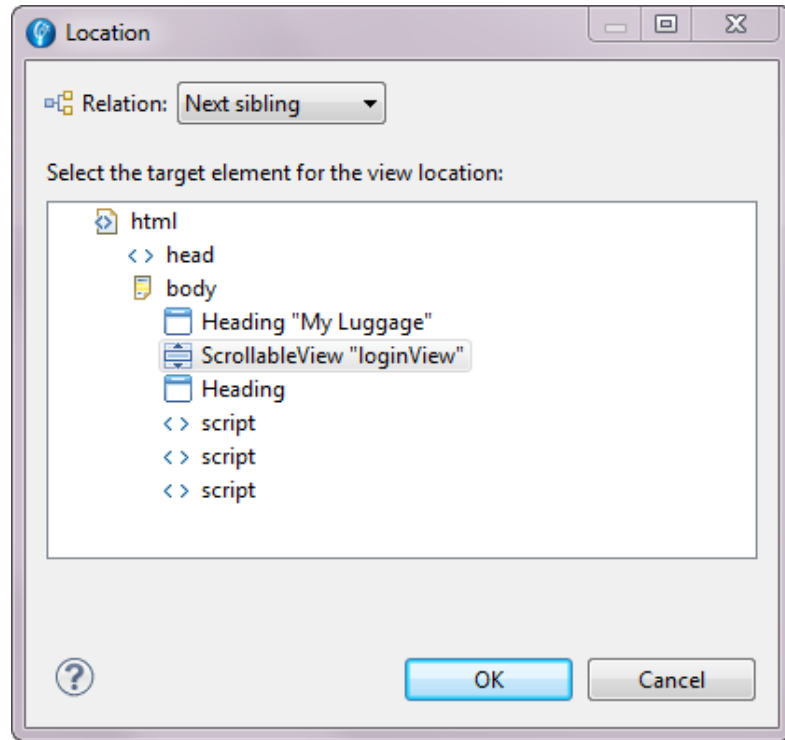


Figure 5-37 Specifying the location of the new ScrollView

- c. Click **OK**. The location information you specified is now displayed in the Add Dojo Mobile View wizard (Figure 5-38).

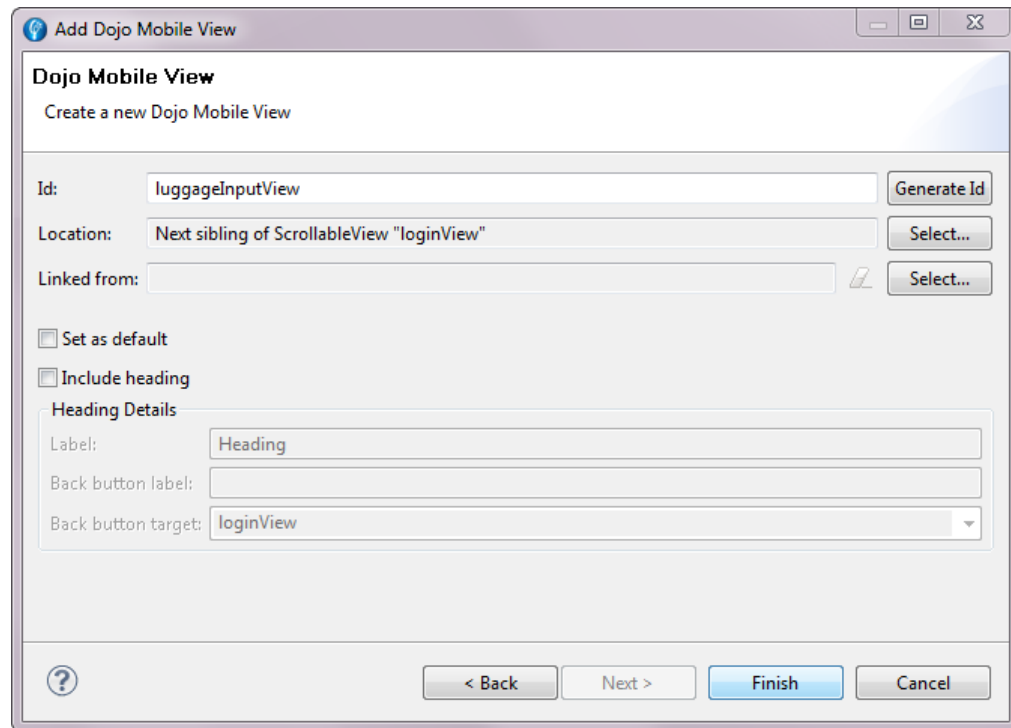


Figure 5-38 Location shown in the Add Dojo Mobile View wizard

- Click **Finish**. The new ScrollableView for the Luggage ID input screen is now displayed in the Mobile Navigation view, as shown in Figure 5-39, and added to the HTML.

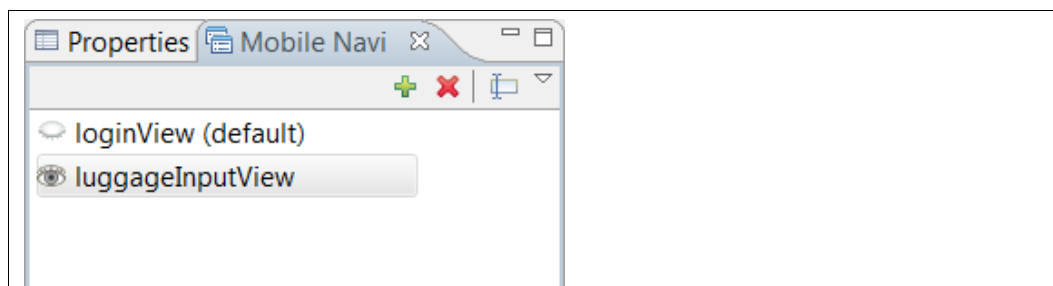


Figure 5-39 Mobile Navigation view showing newly created `luggageInputView`

- Double-click **`luggageInputView`** in the Mobile Navigation view to start laying out the screen in Rich Page Editor.
- Drag the needed additional widgets into the new ScrollableView widget. Then, use the methods described in “Login screen” on page 245 to change each widget’s label text, ID, and properties attributes to what is shown in Table 5-1. The Scan Bar Code button is red, which is defined by the `class=mbIRedButton` property.

Table 5-1 Label text, ID, and properties attributes for Luggage ID input screen widgets

Widget type	Label text	ID	Properties
RoundRect	Not applicable	Not applicable	Not applicable
Text Box	Luggage ID:	<code>luggageTrackingId</code>	<code>width:100%</code>
Button	Scan Bar Code	<code>scanBtn</code>	<code>width:100%</code> <code>class='mbIRedButton'</code>
Button	Retrieve Information	<code>retrieveInformationBtn</code>	<code>width:100%</code>
Button	Logout	<code>logoutBtn</code>	<code>width:100%</code>

The Design view in Rich Page Editor now shows the completed Luggage ID input screen, as shown in Figure 5-40 on page 253. The body section of the HTML source look like what is shown in Example 5-9 on page 253.

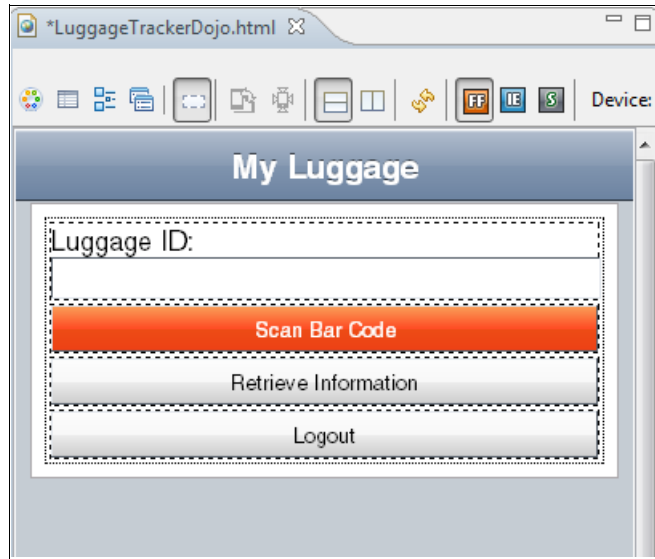


Figure 5-40 Completed Luggage ID input screen displayed in Rich Page Editor

Example 5-9 ScrollableView section of HTML source for the Luggage ID input screen

```
<div data-dojo-type="dojox.mobile.ScrollableView" id="luggageInputView">
  <div data-dojo-type="dojox.mobile.RoundRect">
    <table style="width:100%">
      <tr><td><label for="luggageTrackingId">Luggage ID:</label>
        <input data-dojo-type="dojox.mobile.TextBox" id="luggageTrackingId"
style="width:100%">
      </td></tr>
      <tr><td><button data-dojo-type="dojox.mobile.Button" id="scanBtn"
style="width:100%" class="mblRedButton">Scan Bar Code</button>
      </td></tr>
      <tr><td><button data-dojo-type="dojox.mobile.Button"
id="retrieveInformationBtn" style="width:100%">Retrieve Information</button>
      </td></tr>
      <tr><td><button data-dojo-type="dojox.mobile.Button" id="logoutBtn"
style="width:100%">Logout</button>
      </td></tr>
    </table>
  </div>
</div>
```

Luggage information screen

You can create the Luggage information screen using the same basic steps used for the two screens just described. Start by creating a ScrollableView named `luggageInformationView`, then add the needed widgets and change the text, ID, and properties of each widget. The required widgets along with their label text, ID, and properties values are listed in Table 5-2.

Table 5-2 Label text, ID, and, and properties values for Luggage information screen widgets

Widget type	Label text	ID	Properties
RoundRect	N/A	N/A	N/A
TextBox	Current Status	<code>luggageStatus</code>	<code>width:100%</code>
TextBox	Owner Name	<code>luggageOwner</code>	<code>width:100%</code>
TextBox	Current Location	<code>luggageCurrentLocation</code>	<code>width:100%</code>
TextBox	Next Destination	<code>luggageNextDestination</code>	<code>width:100%</code>
TextBox	Final Destination	<code>luggageFinalDestination</code>	<code>width:100%</code>
TextBox	Last Update	<code>luggageLastUpdate</code>	<code>width:100%</code>
TextArea	Comments	<code>luggageComments</code>	<code>width:100%</code>
Button	Change Delivery Address	<code>changeDeliveryAddressButton</code>	<code>width:100%</code>
Button	Close	<code>closeLuggageInformationViewBtn</code>	<code>width:100%</code> <code>class="mbIRed Button"</code>

The TextBox fields on this screen are for display only, so for each TextBox widget, set the Initial state to **Read-only** on the Tag tab of the Properties view.

When finished, your display looks similar to Figure 5-41 on page 255 and the scrollableView in the HTML source looks like the source shown in Example 5-10 on page 255.

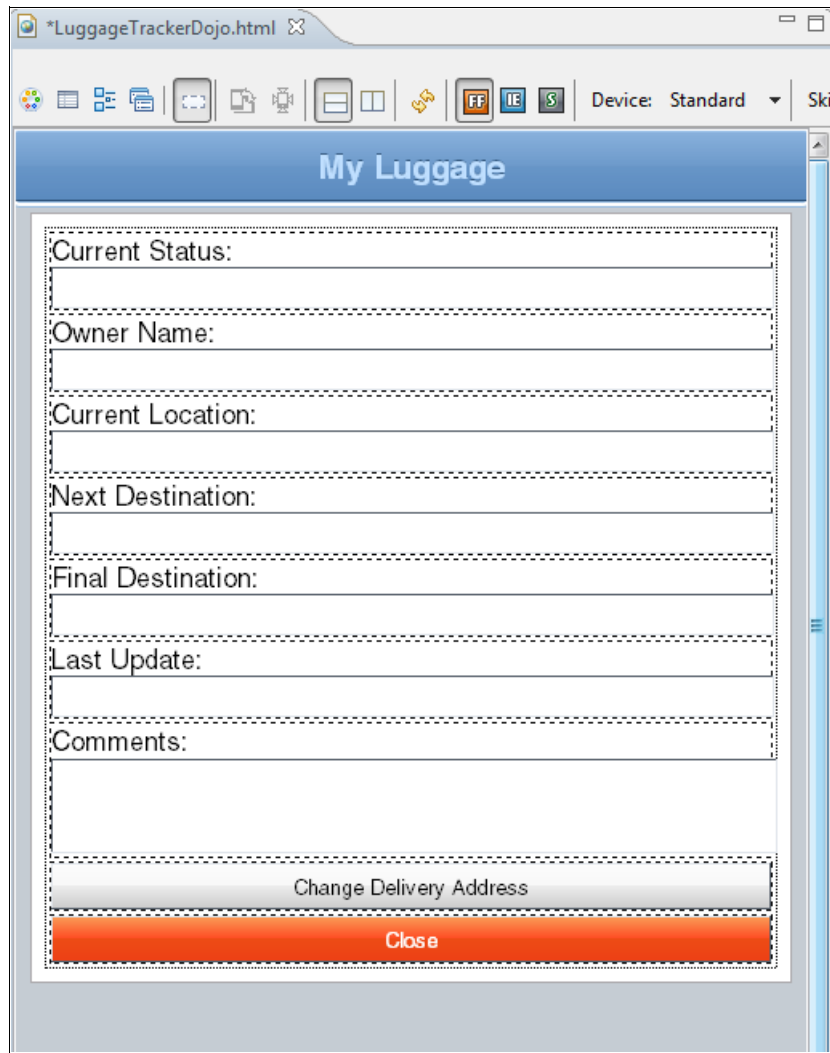


Figure 5-41 Luggage information screen displayed in Rich Page Editor

Example 5-10 ScrollableView section of HTML source for the Luggage information screen

```
<div data-dojo-type="dojox.mobile.ScrollableView" id="luggageInformationView">
  <div data-dojo-type="dojox.mobile.RoundRect">
    <table style="width:100%">
      <tr><td><label for="luggageStatus">Current Status:</label>
        <input id="luggageStatus" data-dojo-type="dojox.mobile.TextBox"
style="width:100%" readonly></td></tr>
      <tr><td><label for="luggageOwner">Owner Name:</label>
        <input id="luggageOwner" data-dojo-type="dojox.mobile.TextBox"
style="width:100%" readonly></td></tr>
      <tr><td><label for="luggageCurrentLocation">Current Location:</label>
        <input id="luggageCurrentLocation" data-dojo-type="dojox.mobile.TextBox"
style="width:100%" readonly></td></tr>
      <tr><td><label for="luggageNextDestination">Next Destination:</label>
        <input id="luggageNextDestination" data-dojo-type="dojox.mobile.TextBox"
style="width:100%" readonly></td></tr>
      <tr><td><label for="luggageFinalDestination">Final Destination:</label>
        <input id="luggageFinalDestination" data-dojo-type="dojox.mobile.TextBox"
style="width:100%" readonly></td></tr>
    </table>
    <div>Change Delivery Address</div>
    <div>Close</div>
  </div>
</div>
```

```

        <tr><td><label for="luggageLastUpdate">Last Update:</label>
        <input id="luggageLastUpdate" data-dojo-type="dojox.mobile.TextBox"
style="width:100%" readonly></td></tr>
        <tr><td><label for="luggageComments">Comments:</label>
        <textarea id="luggageComments" data-dojo-type="dojox.mobile.TextArea"
style="width:100%" readonly></textarea></td></tr>
        <tr><td><button data-dojo-type="dojox.mobile.Button"
id="changeDeliveryAddressBtn" style="width:100%">Change Delivery Address</button>
        <tr><td><button data-dojo-type="dojox.mobile.Button"
id="closeLuggageInformationViewBtn" class="mb1RedButton"
style="width:100%">Close</button>
    </table>
</div>
</div>
</div>

```

Luggage delivery address screen

You can create the Luggage delivery address screen using the same basic steps used for the other screens. Start by creating a ScrollableView named `luggageDeliveryView`. Then, add the needed widgets, and change the text, ID, and properties of each widget. The required widgets and their label text, ID, and properties values are listed in Table 5-3.

Table 5-3 Label text, ID, and, and properties values for Luggage delivery address screen widgets

Widget type	Label text	ID	Properties
TextBox	Luggage ID	<code>luggageDeliveryId</code>	<code>width:100%</code>
TextBox	Name	<code>luggageDeliveryOwner</code>	<code>width:100%</code>
TextBox	Address Line 1	<code>luggageDeliveryAddress1</code>	<code>width:75%</code>
Button	Map	<code>hotelMapBtn</code>	<code>width:20%</code>
TextBox	Address Line 2	<code>luggageDeliveryAddress2</code>	<code>width:100%</code>
TextBox	City	<code>luggageDeliveryCity</code>	<code>width:100%</code>
TextBox	State	<code>luggageDeliveryState</code>	<code>width:100%</code>
TextBox	Zip Code	<code>luggageDeliveryZipCode</code>	<code>width:100%</code>
TextBox	Phone Number	<code>luggageDeliveryPhoneNumber</code>	<code>width:100%</code>
TextArea	Comments	<code>luggageDeliveryComments</code>	<code>width:100%</code>
Button	Save and return	<code>updateDeliveryBtn</code>	<code>width:100%</code>
Button	Cancel	<code>cancelDeliveryBtn</code>	<code>width:100%</code> <code>class="mb1RedButton"</code>

Note that Address Line 1 and the Map button will be within the same row of the table.

When finished, your display looks similar to Figure 5-42 on page 257; the scrollableView in the HTML source looks like the source shown in Example 5-11 on page 257.

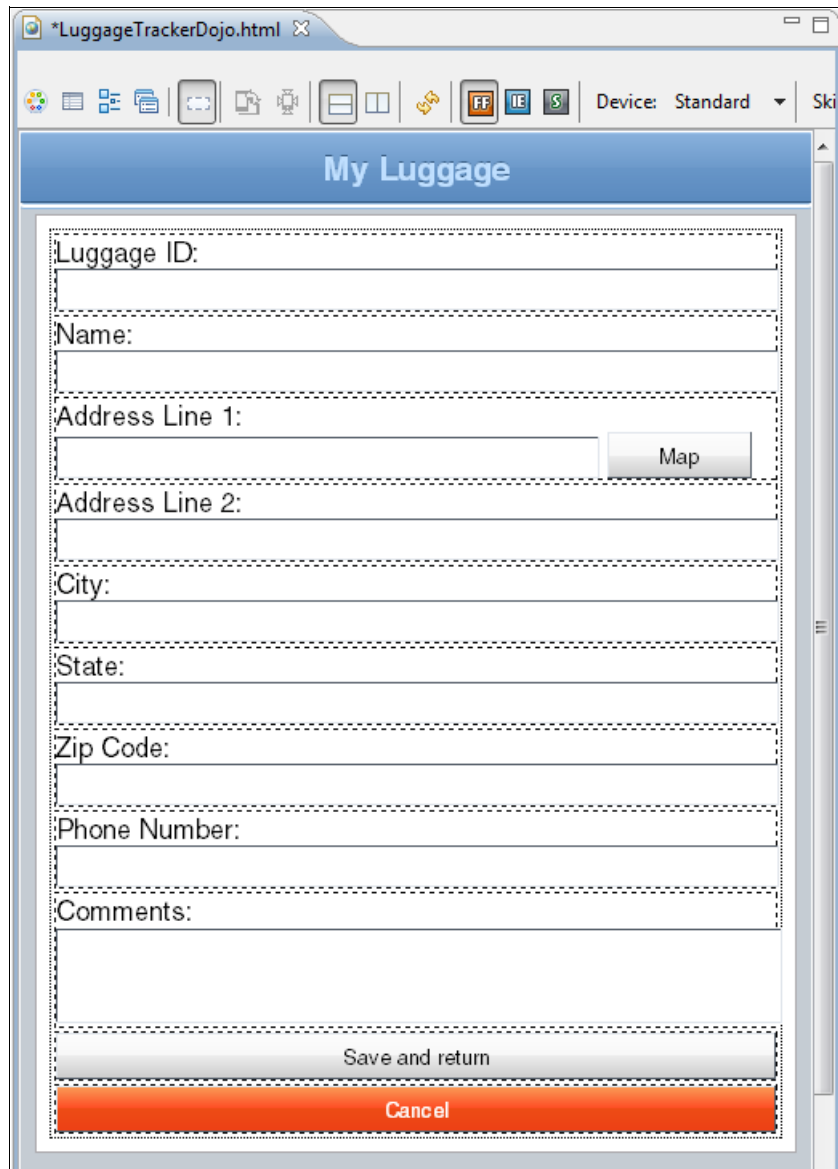


Figure 5-42 Luggage delivery address screen displayed in Rich Page Editor

Example 5-11 ScrollableView section of HTML source for the Luggage delivery address screen

```
<div data-dojo-type="dojox.mobile.ScrollableView" id="luggageDeliveryView">
  <div data-dojo-type="dojox.mobile.RoundRect">
    <table style="width:100%;">
      <tr><td><label for="luggageDeliveryId">Luggage ID:</label>
        <input id="luggageDeliveryId" data-dojo-type="dojox.mobile.TextBox"
style="width:100%;">
      </td></tr>
      <tr><td><label for="luggageDeliveryOwner">Name:</label>
        <input id="luggageDeliveryOwner"
data-dojo-type="dojox.mobile.TextBox" style="width:100%;">
      </td></tr>
      <tr><td><label for="luggageDeliveryAddress1">Address Line 1:</label><br/>
        <input id="luggageDeliveryAddress1"
data-dojo-type="dojox.mobile.TextBox" style="width:75%;">
    </table>
  </div>
  <div>
    <button>Save and return</button>
    <button>Cancel</button>
  </div>
</div>
```

```

        <button id="hotelMapBtn" data-dojo-type="dojox.mobile.Button"
style="width:20%;">Map</button>
    </td></tr>
    <tr><td><label for="luggageDeliveryAddress2">Address Line 2:</label>
        <input id="luggageDeliveryAddress2"
data-dojo-type="dojox.mobile.TextBox" style="width:100%;">
    </td></tr>
    <tr><td><label for="luggageDeliveryCity">City:</label>
        <input id="luggageDeliveryCity"
data-dojo-type="dojox.mobile.TextBox" style="width:100%;">
    </td></tr>
    <tr><td><label for="luggageDeliveryState">State:</label>
        <input id="luggageDeliveryState"
data-dojo-type="dojox.mobile.TextBox" style="width:100%;">
    </td></tr>
    <tr><td><label for="luggageDeliveryZipCode">Zip Code:</label>
        <input id="luggageDeliveryZipCode"
data-dojo-type="dojox.mobile.TextBox" style="width:100%;"></input>
    </td></tr>
    <tr><td><label for="luggageDeliveryPhoneNumber">Phone Number:</label>
        <input id="luggageDeliveryPhoneNumber"
data-dojo-type="dojox.mobile.TextBox" style="width:100%;"></input>
    </td></tr>
    <tr><td><label for="luggageDeliveryComments">Comments:</label>
        <textarea id="luggageDeliveryComments"
data-dojo-type="dojox.mobile.TextArea" style="width:100%;"></textarea>
    </td></tr>
    <tr><td><button id="updateDeliveryBtn"
data-dojo-type="dojox.mobile.Button" style="width:100%;">Save and return</button>
    <tr><td><button id="cancelDeliveryBtn"
data-dojo-type="dojox.mobile.Button" class="mb1RedButton"
style="width:100%;">Cancel</button>
    </td></tr>
</table>
</div>
</div>

```

The final HTML, which contains all of the screens that you just created, can be found in “Source for LuggageTrackerDojo.html” on page 360.

Connecting the buttons

After you create the screens for the user interface by using Dojo Mobile, you connect the buttons to the JavaScript so that when the user clicks them, the correct functions are performed. Because MyLuggage provides many of the same functions as LuggageTracker, most of the back-end services are already available. Most of the buttons are connected in the following sections. The Map button is connected to the client-side JavaScript functions in 5.4.3, “Integrating the map into the mobile application” on page 274; the Login button is connected to the client-side JavaScript functions in 5.4.4, “Adding the new authentication mechanism” on page 281.

Scan Bar Code, Logout, and Retrieve Information buttons

The functions to handle the Scan Bar Code, Logout, and Retrieve Information buttons on the Luggage ID input screen are included in a new file, called `Search.js`. To create this new file with the functions, complete these steps:

1. Expand the `CompanyAirlines` project in Worklight Studio.
2. Expand the `apps/common/js` folder of the `LuggageTrackerDojo` application.
3. Create a new file named `Search.js` by right-clicking the `js` folder and selecting **New** → **JavaScript Source File**. Enter the name `Search.js` into the dialog box and click **Finish**.
4. Add the `dojo/on` and `dijit/registry` modules, which handle mobile application events, by adding the following lines into the `Search.js` file:

```
define(["dojo/on", "dijit/registry", "dojox/mobile/ProgressIndicator"],
        function(on, registry, indicator){
```
5. Add the JavaScript to call the bar code scanner when the Scan Bar Code button is clicked. This is done by adding the JavaScript shown in Example 5-12 to the file.

Example 5-12 JavaScript source to call the bar code scanner

```
function scanBtnClicked(){
    if (window.plugins && window.plugins.barcodeScanner) {
        window.plugins.barcodeScanner.scan(function(result) {
            if (!result.cancelled) {
                registry.byId("luggageTrackingId").set("value", result.text);
            }
        }, function(error) {
            alert(error);
        });
    } else {
        alert("No Scanner available");
    }
};
```

6. Add the JavaScript to call the business services adapter to retrieve the luggage information. This is done by appending the JavaScript, in Example 5-13, to the file.

Example 5-13 JavaScript source to retrieve the luggage information using the adapter

```
function retrieveInformationBtnClicked(){
    var id = registry.byId("luggageTrackingId").get("value");
    var prog = indicator.getInstance();
    prog.start();
    WL.Client.invokeProcedure({
        adapter: "BusinessServicesAdapter",
        procedure: "getLuggageInformation",
        parameters: [id]
    }, {
        onSuccess: function(response){
            prog.stop();
            var luggage = response.invocationResult.luggage;
            updateLuggageInformationView(luggage);

            // Delivery address Info
            var address = luggage.deliveryAddress;
            updateDeliveryInformationView(id, address);
        }
    });
};
```

```

registry.byId("luggageInputView").performTransition("luggageInformationView",
1, "slide");
    },
    onFailure: function(response){
        prog.stop();
        alert("Failed to connect to the server.");
    }
});
};

function updateLuggageInformationView(luggage){
    registry.byId("luggageStatus").set("value", luggage.status);
    registry.byId("luggageOwner").set("value", luggage.luggageOwner);
    registry.byId("luggageCurrentLocation").set("value",
luggage.currentLocation);
    registry.byId("luggageNextDestination").set("value",
luggage.nextLocation);
    registry.byId("luggageFinalDestination").set("value",
luggage.finalDestination);
    registry.byId("luggageComments").set("value", luggage.comments);
}

function updateDeliveryInformationView(id, address){
    registry.byId("luggageDeliveryId").set("value", id);
    registry.byId("luggageDeliveryOwner").set("value", address.name);
    registry.byId("luggageDeliveryAddress1").set("value",
address.addressLine1);
    registry.byId("luggageDeliveryAddress2").set("value",
address.addressLine2);
    registry.byId("luggageDeliveryCity").set("value", address.city);
    registry.byId("luggageDeliveryState").set("value", address.state);
    registry.byId("luggageDeliveryZipCode").set("value", address.zipCode);
    registry.byId("luggageDeliveryPhoneNumber").set("value",
address.phoneNumber);
    registry.byId("luggageDeliveryComments").set("value", address.comments);
}

```

7. Add the function for the Logout button by appending the JavaScript, in Example 5-14, to the file.

Example 5-14 JavaScript source for the Logout button

```

function logoutBtnClicked(){
    registry.byId("luggageInputView").performTransition("loginView", -1,
"slide");
};

```

8. Connect the buttons to the functions that you just added (in the past several steps) and close the define block by appending the lines in Example 5-15 to the Search.js file.

Example 5-15 JavaScript source to connect the buttons to the correct function when clicked

```
    return {
      initView: function(){
        on(registry.byId("scanBtn"), "click", scanBtnClicked);
        on(registry.byId("retrieveInformationBtn"), "click",
retrieveInformationBtnClicked);
        on(registry.byId("logoutBtn"), "click", logoutBtnClicked);
      }
    };
  });
```

9. Save and close the Search.js file.

The complete Search.js file contents is in “Dojo mobile application source files” on page 360.

Change Delivery Address and Close buttons

The functions to handle the Change Delivery Address and Close buttons on the Luggage information screen are included in a new file, named Information.js. To create this new file with the functions, complete these steps:

1. Create a new file named Information.js by right-clicking the js folder and selecting **New → JavaScript Source File**. Enter the name Information.js into the dialog box and click **Finish**.
2. The Change Delivery Address and Close buttons each change what is displayed on the screen, transitioning users to the Luggage delivery address screen (named luggageDeliveryView) or the Luggage ID input screen (named luggageInputView), respectively. To handle these functions, add the JavaScript shown in Example 5-16 to the new Information.js file.

Example 5-16 JavaScript source to display new screens

```
define(["dojo/on", "dijit/registry"], function(on, registry){
  return {
    initView: function(){
      on(registry.byId("changeDeliveryAddressBtn"), "click", function(){

registry.byId("luggageInformationView").performTransition("luggageDeliveryView"
, 1, "slide");
      });

      on(registry.byId("closeLuggageInformationViewBtn"), "click",
function(){

registry.byId("luggageInformationView").performTransition("luggageInputView",
-1, "slide");
      });
    }
  };
});
```

3. Save and close the Information.js file.

The contents of the `Information.js` file can be found in “Dojo mobile application source files” on page 360.

Cancel and Save and return buttons

The functions to handle the Save and return and Cancel buttons on the Change delivery address screen are included in a new file, called `Delivery.js`. To create this new file with the functions, complete the following steps:

1. Create a new file named `Delivery.js` by right-clicking the `js` folder and selecting **New** → **JavaScript Source File**. Enter the name `Delivery.js` into the dialog box and click **Finish**.
2. Add the JavaScript in Example 5-17 to the new `Delivery.js` file. This JavaScript sends the address information to the adapter to be saved, handles the success and failure conditions, and performs the necessary screen transitions.

Example 5-17 JavaScript source to call the adapter to update the delivery address

```
define(["dojo/on", "dijit/registry"], function(on, registry){
    function getUpdatedInformation(){
        var luggageId = registry.byId("luggageDeliveryId").get("value");
        var name = registry.byId("luggageDeliveryOwner").get("value");
        var addressLine1 = registry.byId("luggageDeliveryAddress1").get("value");
        var addressLine2 = registry.byId("luggageDeliveryAddress2").get("value");
        var city = registry.byId("luggageDeliveryCity").get("value");
        var state = registry.byId("luggageDeliveryState").get("value");
        var zipCode = registry.byId("luggageDeliveryZipCode").get("value");
        var phoneNumber =
registry.byId("luggageDeliveryPhoneNumber").get("value");
        var comments = registry.byId("luggageDeliveryComments").get("value");
        return [luggageId, name, addressLine1, addressLine2, city, state,
zipCode, phoneNumber, comments];
    };

    function updateDeliveryInformation(){
        var params = getUpdatedInformation();
        var invocationData = {
            adapter : 'BusinessServicesAdapter',
            procedure : 'addLuggageDeliveryInformation',
            parameters : params
        };
        WL.Client.invokeProcedure(invocationData, {
            onSuccess : function(){
                alert("Update complete.");
            }
        });

        registry.byId("luggageDeliveryView").performTransition("luggageInformationView"
, -1, "slide");
    },
    onFailure : function(){
        alert("Failed to connect to server.");
    },
    });
};

return {
    initView: function(){
        on(registry.byId("updateDeliveryBtn"), "click",
updateDeliveryInformation);
    }
};
```



```

        on(registry.byId("cancelDeliveryBtn"), "click", function(){

registry.byId("luggageDeliveryView").performTransition("luggageInformationView"
, -1, "slide");
        });
    }
};
});

```

3. Save and close the Delivery.js file.

The contents of the Delivery.js file is in “Dojo mobile application source files” on page 360.

Registering the packages in the dojoConfig

Next, define the module loading path for application modules. Dojo follows the Asynchronous Module Loading (AMD) specification to load JavaScript modules. The paths of the Dojo libraries are predefined, but you still must add the path of the custom application module that was written for the mobile application.

To register the packages in the dojoConfig, use the following steps:

1. Open the LuggageTrackerDojo.html file, located in the common folder of the LuggageTrackerDojo application.
2. Near the top of the file, locate the <script> tag that contains the src=dojo/dojo.js attribute, and delete the data-dojo-config attribute from the tag.
3. Add the JavaScript in Example 5-18 to the file, placing it immediately before the <script> tag that contains the src=dojo/dojo.js attribute.

Example 5-18 JavaScript to add the custom application module

```

<script type="text/javascript">
    dojoConfig = {
        parseOnLoad: true,
        isDebug: false,
        async: true,
        baseUrl: "",
        packages:[
            { "name": "dojo", "location": "dojo" },
            { "name": "dijit", "location": "dijit" },
            { "name": "dojox", "location": "dojox" },
            { "name": "luggage", "location" : "js" }
        ]
    };
</script>

```

4. Save and close the LuggageTrackerDojo.html file.

Finally, you must update the initialization of dojo to import the specific custom application modules, and to initialize each of the screens:

1. Open the LuggageTrackerDojo.js file located under the common/js folder of LuggageTrackerDojo application.
2. Locate the dojoInit function near the bottom of the file and replace it with the JavaScript from Example 5-19 on page 264.

Example 5-19 JavaScript to register each file, function, and initialize the screens

```
<function dojoInit() {
    require([ "dojo", "dijit/registry", "dojo/parser", "dojox/mobile",
"dojox/mobile/compat",
        "dojox/mobile/deviceTheme", "dojox/mobile/ScrollableView",
        "dojox/mobile/Heading", "dojox/mobile/View",
        "dojox/mobile/TextBox", "dojox/mobile/Button",
        "dojox/mobile/RoundRect", "dojox/mobile/TextArea" ],
    function(dojo, registry) {
        dojo.ready(function() {
            require(["luggage/Login", "luggage/Search",
                "luggage/Information", "luggage/Delivery",
                "luggage/Auth"],
                function(login, search, information, delivery, auth){
                    login.initView();
                    search.initView();
                    information.initView();
                    delivery.initView();
                    auth.init();
                });
        });
        window.registry = registry;
    });
};
```

3. Save and close the LuggageTrackerDojo.js file.

5.4.2 Developing the native map page

MyLuggage integrates a native map component that will be used to display hotels near the users location. Because this is a native function, a native map page must be added to a customized Worklight Shell so that it can be used by the application. This native page is written to be platform-dependent using Java for Android. Worklight provides a way to easily navigate between web and native pages and also share data between them.

Integrating a native page has three major advantages:

- ▶ Adds features to your mobile application that cannot be implemented with web technologies
- ▶ Allows you to reuse components of existing native mobile applications
- ▶ Overcomes user perceptions of slow response times when using a non-native map in a hybrid application on Android

For the MyLuggage application, a native Android page will be created to display a map with nearby hotels. When the user clicks on a hotel, the native page returns the hotel's address to the hybrid application, which then displays the address to the user in the Luggage delivery address screen.

The steps to create the native page using a Worklight shell are similar to the steps used to create the bar code scanner using a Worklight shell, described in 4.8.2, "Creating the shell for the bar code scanner" on page 121. There are some differences, however, because MyLuggage support only Android, which means there is only a single platform and no common interface.

Adding the sources to the test application

For the native map component, you use the same Worklight Shell project created in 4.8.2, “Creating the shell for the bar code scanner” on page 121.

Because of the work that was done previously in the same project, the test application of the shell project calls the bar code scanner for LuggageTracker. Because this test application will be reused to test the native page with Google Maps, the test code that calls the bar code scanner must be commented out:

1. Start Worklight Studio if it is not already started.
2. In Project Explorer, expand the WLAirlineShell project until the files under the apps/AirlineShellComponentTest/common/js folder are visible.
3. Open the AirlineShellComponentTest.js file in Rich Page Editor by double-clicking on the file in Project Explorer.
4. Comment out the existing JavaScript code within the wlCommonInit function.
5. Add the test code that will invoke the native map page (which will be created later) by adding the JavaScript functions wlEnvInit, openNativePage, and backFromNativePage, shown in Example 5-20 to the AirlineShellComponentTest.js file.

Also add a call to the openNativePage function within the wlEnvInit function. This function contains information for two hotels, including their longitude and latitude positions and their name and address information. This data is for testing only and is used to indicate hotel locations when the native page is invoked. The openNativePage function takes three parameters:

- The class name of the Android activity to be invoked
- The callback, backFromNativePage, used after the activity ends and returns its result
- The parameters supplied to the Android activity, which for the native map is the hotel information

Example 5-20 JavaScript source for native page functions

```
function wlEnvInit(){
    wlCommonInit();
    // Environment initialization code goes here
    openNativePage();
}

function openNativePage () {
    var hotels = "{ hotels: [ " +
        "{ position: [ 31.260, 121.469 ], title: \"Hotel 1\",
address: \"Address Hotel 1\" }, " +
        "{ position: [ 31.20555, 121.59086 ], title: \"Hotel 2\",
address: \"Address Hotel 2\" } ] }";
    var params = {positions: hotels };
    WL.NativePage.show("com.mapview.HotelMapView", backFromNativePage,
params);
}

function backFromNativePage (data) {
    alert("Received ID: "+ data.HotelIndexID);
}
```

6. Save and close the `AirlineShellComponentTest.js` file.
7. Rebuild the generated Android project by right-clicking on the following file and selecting **Run As → 2 Build Environment and Deploy:**
`apps/AirlineShellComponentTest/android`

The hybrid application is ready to invoke a native page.

Implementing the native map for Android

The map is a native component and, therefore, must have a unique implementation for each platform. The following tasks are required for completing the Android native map implementation:

- ▶ Adding the required sources and resources to the generated native project
- ▶ Creating the class for the hotel information
- ▶ Creating the layout to display the map
- ▶ Creating the activity class
- ▶ Adding the proper permissions and the Google Maps API key
- ▶ Making the changes to the Android template files

Adding the required sources and resources to the generated native project

Currently, the Android test project attempts to invoke a native page, which is not yet implemented. Because the native page will display a Google Map, additional libraries must be added to the test project:

1. Start Worklight Studio if it is not already started.
2. In Project Explorer, right-click the generated Android project, `WLAirlineShellAirlineShellComponentTestAndroid`, and select **Android Tools → Add Support Library**.
3. In the Choose Packages to Install dialog, read and accept the license terms and click **Install**.
4. Right-click the generated Android project again in Project Explorer and select **Properties**.
5. In the Properties dialog for the project, select **Android** from the left navigation menu.
6. Click **Add** to add a new library.
7. Select **google-play-services_lib**, which you already installed, from the list, and click **OK**.
8. The Properties dialog now shows the `google-play-services_lib` library in the table. Click **OK**.

The required native map source is now added to the generated Android project.

Creating the class for the hotel information

The map for `MyLuggage` will display hotel information about a map, and a new class for holding this hotel information must be created. To accomplish this, use the following steps:

1. In Project Explorer, right-click the generated Android project and select **New → Class**.
2. In the New Java Class dialog, enter `com.mapview` for the Package and `Hotel` for the Name and click **Finish**.
3. The Java source file is created and opens in the editor, with the package and class definitions already added. Next, add the source code shown in Example 5-21 on page 267, replacing all of the lines that were generated when the file was created.

The `Hotel` class contains two instance variables, which are used to store the name and position of the hotel. The class also contains the constructor and getter/setter methods.

Example 5-21 Source of the hotel class

```
package com.mapview;

import com.google.android.gms.maps.model.LatLng;

public class Hotel {
    private String name;
    private LatLng position;

    public Hotel(String name, double latitude, double longitude) {
        this.name = name;
        this.position = new LatLng(latitude, longitude);
    }

    public String getName() {
        return name;
    }

    public LatLng getPosition() {
        return position;
    }
}
```

4. Save and close the `Hotel.java` file.

Creating the layout to display the map

A layout file defines the contents of the device screen, similar to how the `ScrollView` defined the contents of the device screen in the Dojo-based user interface. The layout for the native map will display a Google map with hotels, so the layout will contain a single `SupportMapFragment` element to display the map.

To create the layout, use the following steps:

1. In Project Explorer, right-click the generated Android project and select **New** → **Other**.
2. In the New dialog, expand the `android` folder and select **Android** → **Android XML Layout File** to use the Android layout XML file wizard and then click **Next**.
3. Enter `activity_hotel_map_view` for the name for the layout file and then click **Finish**, which creates the XML file and opens the file in Rich Page Editor.
4. Switch to the XML source by clicking on the tab that contains the file named `activity_hotel_map_view.xml`.
5. Replace the existing XML with the XML shown in Example 5-22, which defines that the Google map fills the entire screen.

Example 5-22 XML source of the layout to display the map

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    class="com.google.android.gms.maps.SupportMapFragment"/>
```

6. Save and close the `activity_hotel_map_view.xml` file.

Creating the activity class

Next, create the activity that displays a Google Map. An *activity* controls the screen that is shown on the device, including displaying the user interface components and responding to actions on behalf of the operating system (such as pausing when there is an incoming call). To create the corresponding activity class in the Android manifest file, use the following steps:

1. In Project Explorer, open the `AndroidManifest.xml` file that is at the root of the generated Android project.
2. Select the **Application** tab to display the Android Manifest Application.
3. At the bottom of the page, there is a table containing Application Nodes. Click **Add** to add a new application node.
4. In the dialog that is displayed, select **Activity** and then click **OK**.
5. The dialog closes and the Attributes for Activity section is displayed to the right of the Application Nodes table, as shown in Figure 5-43, listing the attributes for the activity.

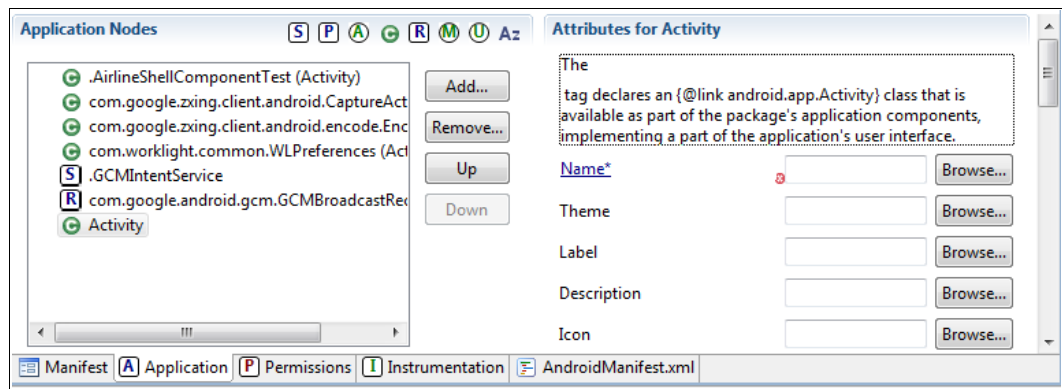


Figure 5-43 Newly created Activity displayed in the Android Manifest Application

6. Click the **Name** link in the Attributes for Activity section. The New Java Class dialog opens.
7. In the New Java Class dialog, enter `com.mapview` in the Package field and `HotelMapView` in the Name field, and then click **Finish**. This creates a new Java class named `HotelMapView.java` and opens it in Rich Page Editor. In the activity attributes (Figure 5-43), the package and class name of this new Java class are automatically entered into the Name field.
8. The newly created Java file in Rich Page Editor contains some auto-generated Java code. Replace this auto-generated code with the Java code shown in Example 5-23.

This activity contains two methods:

- The `onCreate` method is called during the `WL.NativePage.show` invocation, which is part of the `openNativePage` function that was added to `AirlineShellComponentTest.js` (see Figure 5-11 on page 222). This method creates a marker (with an index) for each hotel displayed on the map. The index is used when the user clicks on a hotel to select it.
- The `onInfoWindowClick` method is called when the user clicks on a marker (equating to a hotel) on the map. It returns the index of the selected hotel to the application.

Example 5-23 Java source for the `HotelMapView` activity

```
package com.mapview;
import java.util.ArrayList;
import java.util.HashMap;
```

```

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.util.Log;

import com.AirlineShellComponentTest.R;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.GoogleMap.OnInfoWindowClickListener;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

public class HotelMapView extends FragmentActivity implements
    OnInfoWindowClickListener {

    private static final String TAG_HOTELS = "hotels";
    private static final String TAG_HOTEL_NAME = "title";
    private static final String TAG_HOTEL_POSITION = "position";
    private static final String TAG = "HotelMapView";

    private HashMap<String, Integer> markerHotelCorrelation;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // get the json containing the hotel information
        String jsonString = getIntent().getStringExtra("positions");

        // parse the json and store the hotel information in this arrayList
        ArrayList<Hotel> hotels = null;
        try {
            JSONObject json = new JSONObject(jsonString);
            JSONArray jsonHotels = json.getJSONArray(TAG_HOTELS);
            hotels = new ArrayList<Hotel>();

            for (int i = 0; i < jsonHotels.length(); i++) {
                JSONObject jsonHotel = jsonHotels.getJSONObject(i);

                String name = jsonHotel.getString(TAG_HOTEL_NAME);
                JSONArray position =
                    jsonHotel.getJSONArray(TAG_HOTEL_POSITION);

                double latitude = position.getDouble(0);
                double longitude = position.getDouble(1);

                Hotel hotel = new Hotel(name, latitude, longitude);
                hotels.add(hotel);
            }
        } catch (JSONException e) {

```

```

        Log.e(TAG, "Problem dealing with the provided JSON String", e);
    }

    setContentView(R.layout.activity_hotel_map_view);

    if (!hotels.isEmpty()) {
        GoogleMap map =
            ((SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map)).getMap();

        //in case the Google-Play-Services are not installed or not
        //up-to-date the map will be null
        if (map != null) {
            map.setMapType(GoogleMap.MAP_TYPE_NORMAL);

            //for each hotel create a marker and store the id
            //for simplicity the id is just the index of the hotel in
            //the hotel list
            markerHotelCorrelation = new HashMap<String, Integer>();
            for (int i = 0; i < hotels.size(); i++) {
                Hotel hotel = hotels.get(i);
                Marker marker = map.addMarker(new
                MarkerOptions().position(hotel.getPosition()).title(hotel.getName()));
                markerHotelCorrelation.put(marker.getId(), i);
            }
        }
        map.setOnInfoWindowClickListener(this);

        map.animateCamera(CameraUpdateFactory.newLatLngZoom(hotels.get(0).getPosition(), 10));
    }
}

@Override
public void onInfoWindowClick(Marker marker) {
    // get the index of the selected hotel
    int indexID = markerHotelCorrelation.get(marker.getId());

    // create Intent to add the index of the selected hotel to
    Intent resultIntent = new Intent();
    resultIntent.putExtra("HotelIndexID", Integer.valueOf(indexID));

    // set the result and return back to the caller
    setResult(RESULT_OK, resultIntent);
    finish();
}
}

```

9. Save and close the HotelMapView.java file.

10. Save the AndroidManifest.xml file but do not close it, because you make additional changes in the next section.

Adding the proper permissions and the Google Maps API key

All of the code has now been created to display a map with the two test hotels and retrieve information for a selected hotel. Now, add your Google Maps API key, and the proper permissions for displaying the map, into the Android manifest file.

Use the following steps:

1. In the `AndroidManifest.xml` file, click the **AndroidManifest.xml** tab, to show the XML source for the manifest file.
2. To add the proper permissions for Google Maps, start by locating the comment line that contains `<!-- Barcode Scanner permission -->` and the two lines that contain `<uses-permission>`, which were added for the bar code scanner. Immediately beneath these lines, add the lines shown in Example 5-24.

Example 5-24 Permissions for Google Maps

```
<!-- Google Maps permissions -->
<permission android:name="com.AirlineShellComponent.permission.MAPS_RECEIVE"
            android:protectionLevel="signature" />

<uses-permission
    android:name="com.AirlineShellComponent.permission.MAPS_RECEIVE" />
<uses-permission
    android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<uses-feature android:glEsVersion="0x00020000" android:required="true" />
```

3. Scroll to the bottom of the `AndroidManifest.xml` file, locate the application closing tag (`</application>`), and then add the following lines, replacing `<YOUR_GOOGLE_MAPS_API_KEY>` with the required Google Maps API key.

```
<meta-data android:name="com.google.android.maps.v2.API_KEY"
            android:value="<YOUR_GOOGLE_MAPS_API_KEY>" />
```

4. Save and close the `AndroidManifest.xml` file.

You can now test the native map page using the test application. To do this, connect an Android device to your developer workstation using a USB cable because an Android application using the Google Play services library can be tested only on a real device and not with an Android emulator.

Launch the test application using the same process described in “Testing by using physical devices” on page 159; you see a map similar to Figure 5-44.



Figure 5-44 Native Google Map displaying two hotel markers

Making the changes to the Android template files

After testing the application successfully, the changes that were made in the generated Android project must be made to the shell template files in the WLAirlineShell project. Follow these steps:

1. In Worklight Studio, copy the `com.mapview` package that contains the `Hotel` and `HotelMapView` classes from the `src` folder (located under the `WLAirlineShellAirlineShellComponentTestAndroid` project) to the `src` folder under `components/AirlineShellComponent/android/native/` in the `WLSHELL` project.
2. Copy the `activity_hotel_map_view.xml` file from the `res/layout` folder (located under the `WLAirlineShellAirlineShellComponentTestAndroid` project) to the `res/layout` folder, under `components/AirlineShellComponent/android/native/` in the `WLSHELL` project.
3. Copy the `android-support-v4.jar` file from the `lib` folder (located under the `WLAirlineShellAirlineShellComponentTestAndroid` project) to the `lib` folder under `components/AirlineShellComponent/android/native/` in the `WLSHELL` project.
4. Ensure the shell uses the `google-play-services_lib` project as follows:
 - a. Open the `project.properties.wltemplate.wluser` file in the `components/AirlineShellComponent/android/native/src` folder.
 - b. Add the following line:

```
android.library.reference.1=../../../../../google-play-services_lib
```

5. Save and close the `project.properties.wltemplate.wluser` file.
6. Open the `AndroidManifest.xml.wltemplate.wluser` file (located in the `components/AirlineShellComponent/android/native/src` folder) and add the lines shown in Example 5-24 on page 271 beneath the two lines that contain the `<uses-permission>` tags that were previously added for the bar code scanner.
7. Scroll to the bottom of the file and locate the application closing tag (`</application>`) and add the following lines, replacing `<YOUR_GOOGLE_MAPS_API_KEY>` with your Google Maps API key:


```
<activity android:name="com.mapview.HotelMapView"></activity>
<meta-data android:name="com.google.android.maps.v2.API_KEY"
            android:value="<YOUR_GOOGLE_MAPS_API_KEY>" />
```
8. Save and close the `AndroidManifest.xml.wltemplate.wluser` file.

As explained in “Making the changes to the Android template files” on page 132, you also must change the import statements for the `R.java` file. The Java files in `AirlineShellComponent` need to import the `R.java` file using a Worklight placeholder for the `packageName`, so that the Worklight build process substitutes the proper package name. Because there is only one file to change for the native map, it is easy to manually make the change using these steps:

1. Rename the `HotelMapView.java` file in the following folder of the `WLAirlineShell` project to `HotelMapView.java.wltemplate.wluser`:


```
components/AirlineShellComponent/android/native/src/com/mapview
```
2. Open the newly renamed `HotelMapView.java.wltemplate.wluser` file and change import `com.AirlineShellComponentTest.R;` to the following line:


```
import ${packageName}.R;
```
3. Save and close the `HotelMapView.java.wltemplate.wluser` file.

You can let Worklight re-create the Android project by right-clicking on the `apps/AirlineShellComponent/android` folder under the `WLAirlineShell` project and choosing **Run as** → **Build Environment and Deploy**. Now you can now test the application again and you should see the same results as before.

Building the Worklight shell

Now that the native map is added to and tested within the Worklight shell component, build the Worklight Shell project to generate a shell file. This shell file will then be provided to the mobile web application developers who will use the APIs that are defined within the shell component as part of their hybrid application.

The same shell project was used for the bar code scanner in the `LuggageTracker` application, so the version number of the shell must be updated to indicate a new version and then the shell file with the new version number must be built. Follow these steps to change the version number and build the shell:

1. Expand the `WLAirline` project and expand the `components` folder.
2. Expand the `AirlineShellComponent` folder and then open the `shell-descriptor.xml` file.
3. On the Design tab, select **Shell "AirlineShellComponent"** to display the details of the shell in the Details section.

4. Change the Version to 2.0, as shown in Figure 5-45.

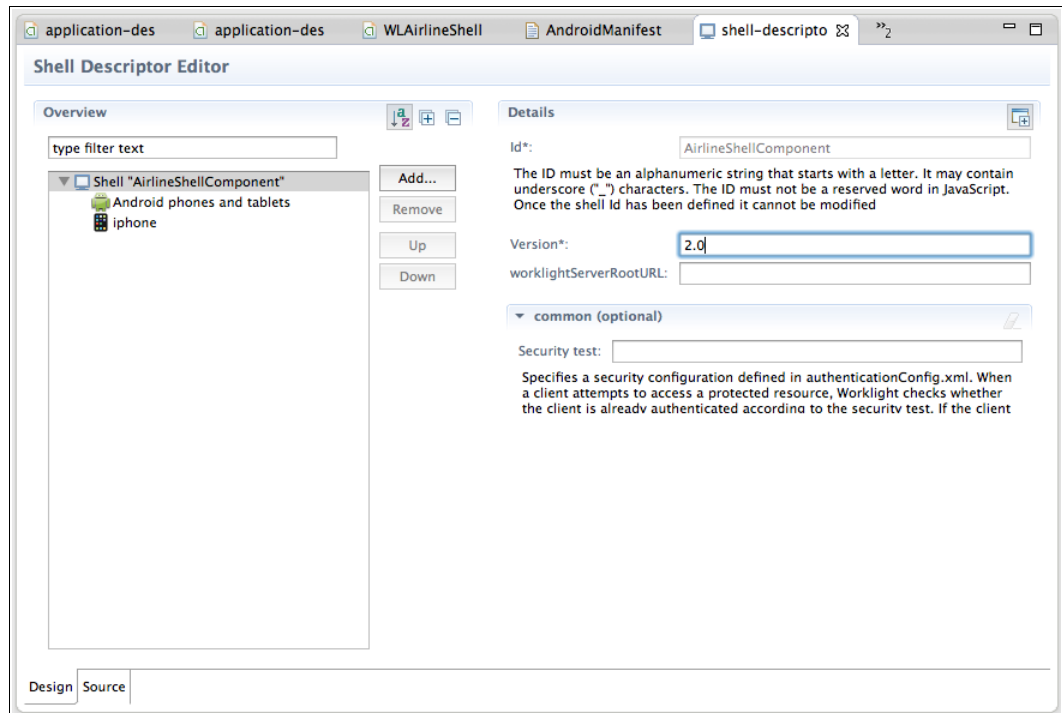


Figure 5-45 Changing the shell version number

5. Save and close the shell-descriptor.xml file.
6. Build the shell by right-clicking the AirlineShellComponent folder and then selecting **Run as** → **1 Build Shell component**. This starts the build process of the shell component.

When the build completes, the new AirlineShellComponent-2.0.wlshell file is in the bin folder of the WLAirlineShell project and is ready to be used in building the application (see 5.4.5, “Building the new mobile application” on page 284).

5.4.3 Integrating the map into the mobile application

In contrast to LuggageTracker, the new MyLuggage application will include a native map to show passengers the hotels that are near their present location. The passenger can select one of these hotels as the luggage delivery location, or find another address on the map. The application and the native map have been developed, so now they must be integrated.

Creating a Worklight environment for Android

The LuggageTrackerDojo application in Worklight Studio currently has only common components. Because the map is a native component, we need a new Worklight environment for Android devices.

To create the Worklight environment for Android devices, use the following steps:

1. Start Worklight Studio if it is not already started.
2. Expand the CompanyAirlines project until the LuggageTrackerDojo folder is visible.

3. Right-click the LuggageTrackerDojo folder and then select **New** → **Worklight Environment**, as shown in Figure 5-46.

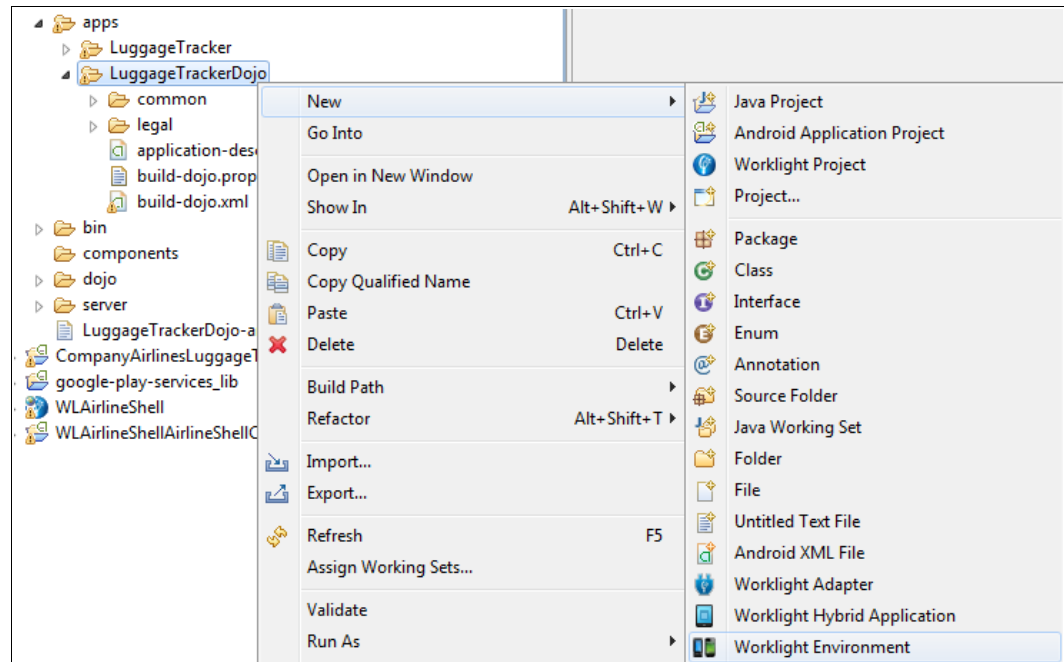


Figure 5-46 Creating a new Worklight environment for Android

4. In the New Worklight Environment dialog, the Project name and Application/Component fields are pre-filled. Because this is an environment for Android, you must select **Android phones and tablets**.
5. Click **Finish**.

A new environment is created under the LuggageTrackerDojo folder. Also created is a new generated Android project named CompanyAirlinesLuggageTrackerDojoAndroid.

Adding the shell to the application

To integrate the native map shell into the application, you must first declare the shell in the application descriptor using these steps:

1. Edit the application-descriptor.xml file in the apps/LuggageTrackerDojo folder.
2. Find the following line:

```
<thumbnailImage>common/images/thumbnail.png</thumbnailImage>
```

Beneath that line, add the following lines:

```
<shell>
  <archive>../../wshell/AirlineShellComponent-2.0.wshell</archive>
</shell>
```

This code defines the location of the wshell file that was created in 5.4.2, “Developing the native map page” on page 264.

3. Save and close the application-descriptor.xml file.
4. Copy the AirlineShellComponent-2.0.wshell file that was created in “Building the Worklight shell” on page 273 from the bin directory of the WLAirlineShell project to the location specified in step 2. If you did not build the shell file, obtain the shell file from your native mobile application developer and place the file in the location specified in step 2.

Adding Google Play services to the application

Similar to how the generated Android test project needed the Google Play Services library to invoke the native map, the actual mobile application needs the same library. For MyLuggage, it will be included as a Support Library.

To add the Google Play services library as a Support Library, use the following steps:

1. Right-click the generated Android project, CompanyAirlinesLuggageTrackerDojoAndroid, and select **Android Tools** → **Add Support Library**.
2. If you have not added the Support Library in Worklight Studio previously, the Choose Packages to Install dialog opens. Read and accept the license terms and then click **Install**.
3. The Android SDK Manager automatically configures the Android Support Library for the project and indicates, in a small dialog, the progress of this task.
4. When the progress dialog is removed, indicating that the support library has been added, right-click the generated Android project again and select **Properties**.
5. In the Properties for CompanyAirlinesLuggageTrackerDojoAndroid, click **Android** in the navigation tree.
6. In the Library group box, click **Add** to add a new library to the project.
7. The Project Selection dialog opens. Select **google-play-services_lib** from the list of available libraries and then click **OK**. The google-play-services_lib project is shown in the table in the Library group box, as shown in Figure 5-47.

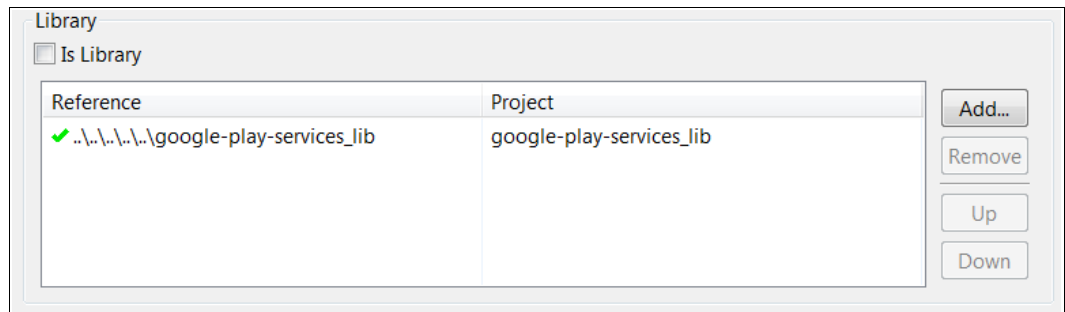


Figure 5-47 The google-play-services_lib added as a Library

8. Click **OK**.

The google-play-services_lib library is now added to the project.

Adding the permissions and API key, and declaring the hotel view

You now need to add the proper permissions for Google Maps, add your Google Maps API key, and define the Activity for the hotel view. This process follows steps that are similar to those completed when testing the map in an earlier subsection.

Use the following steps:

1. In Worklight Studio, open the `AndroidManifest.xml` file in the generated Android project, CompanyAirlinesLuggageTrackerDojoAndroid.
2. To add permissions for Google Maps, add the lines shown in Example 5-25 on page 277 to the file, before the beginning application tag (`<application>`).

Example 5-25 XML to add permissions for Google Maps to the Android manifest file

```
<!-- Google Maps permissions -->
<permission android:name="com.AirlineShellComponent.permission.MAPS_RECEIVE"
            android:protectionLevel="signature" />

<uses-permission
android:name="com.AirlineShellComponent.permission.MAPS_RECEIVE" />
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<uses-feature android:glEsVersion="0x00020000" android:required="true" />
```

3. Now add the source to define your Google Maps API key by adding the following lines immediately above the application closing tag (</application>), and replacing <YOUR_GOOGLE_MAPS_API_KEY> with your Google Maps API key:

```
<activity android:name="com.mapview.HotelMapView"></activity>
<meta-data android:name="com.google.android.maps.v2.API_KEY"
            android:value="<YOUR_GOOGLE_MAPS_API_KEY>" />
```

4. Save and close the AndroidManifest.xml file.

Getting the hotels and showing them on the map

In the Change Delivery Address view, users have the option of showing a map with hotels near their current location by clicking the Map button.

To get the hotel information and display the hotels on the map, use the following steps:

1. Expand the CompanyAirlines project in Worklight Studio.
2. Expand the apps/common/js folder of the LuggageTrackerDojo application.
3. Open the Delivery.js file.
4. Add the JavaScript code shown in Example 5-26 to the file, immediately above the getUpdatedInformation function that was added in “Connecting the buttons” on page 258.

The getHotelList function calls the getHotels procedure of the CastIron adapter, MapAdapter. If the call is successful, the showHotelInMap function then takes the response from the adapter and adds the hotels to the map.

Example 5-26 Functions to get hotels and display them on the map

```
function getHotelList(position){
    var invocationData = {
        adapter : 'MapAdapter',
        procedure : 'getHotels',
        parameters : [ position.coords.latitude, position.coords.longitude]
    };

    WL.Client.invokeProcedure(invocationData, {
        onSuccess : showHotelInMap,
        onFailure : function(){
            alert("Failed to get the hotel list.");
        }
    });
};
```

```
function showHotelInMap(response){
    var hotels = {"hotels" : response.invocationResult.results.items};
    var params = {positions: JSON.stringify(hotels)};
    alert(params.positions);
    WL.NativePage.show("com.mapview.HotelMapView", function(data) {
        if (data && data.HotelIndexID) {
            registry.byId("luggageDeliveryAddress1").set("value",
                hotels.hotels[data.HotelIndexID].vicinity);
        }
    }, params);
};
```

5. Replace the `initView` function in the `Delivery.js` file with the JavaScript shown in Example 5-27, which connects the Map button to the `getHotelList` function.

Example 5-27 JavaScript to connect Map button

```
initView: function(){
    on(registry.byId("updateDeliveryBtn"), "click", updateDeliveryInformation);

    on(registry.byId("cancelDeliveryBtn"), "click", function(){
        registry.byId("luggageDeliveryView").performTransition(
            "luggageInformationView", -1, "slide");
    });

    on(registry.byId("hotelMapBtn"), "click", function(){
        navigator.geolocation.getCurrentPosition(getHotelList);
    });
}
```

6. Save and close the `Delivery.js` file.

Testing the changes

After completing all of the changes necessary to display the map, you can now test the changes by starting the application and running through the anticipated user scenarios. Google Play services are not supported within an Android emulator, so the map functionality in MyLuggage can only be tested using a physical device.

To test the changes using a physical device, use the following steps:

1. Start by connecting a mobile device to the developer's workstation using a USB cable. Ensure that all device drivers are installed and working properly before you deploy the application to the mobile device.
2. Launch the application and log in with a valid user ID and password.
3. Retrieve luggage information using a valid luggage identifier.

4. Click **Change Delivery Address** so that the Luggage delivery address screen is displayed, as shown in Figure 5-48.

Luggage Id:

Name:

Address Line1:

Address Line2:

City:

State:

Zip Code:

Phone Number:

Comments:

Figure 5-48 Luggage delivery address screen showing the new Map button

5. Click **Map**. Your current location is determined and the map displayed, as shown in Figure 5-49.



Figure 5-49 Native map showing nearby hotels

6. Select a hotel by touching the appropriate marker on the map. The selected hotel's address is shown in the Luggage delivery address screen, in Figure 5-50 on page 281.



Figure 5-50 Luggage delivery address screen with selected hotel's address

The Map button's connection to Google Maps and the application's ability to populate the address fields with information for the selected hotel is now tested successfully.

5.4.4 Adding the new authentication mechanism

The authentication model for MyLuggage uses a DataPower XG45 appliance as the security gateway, as first discussed in "Configuring the IBM DataPower XG45 appliance" on page 224. The detailed authentication model is shown in Figure 5-51 on page 282.

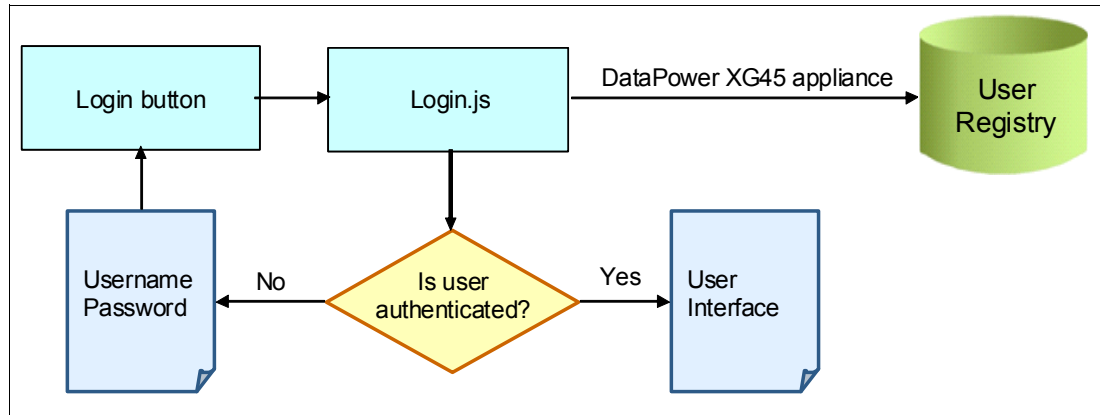


Figure 5-51 Authentication model of the MyLuggage application

In a typical usage scenario, the user clicks the Login button and enters a user name and password for processing by the `Login.js` JavaScript file. These credentials are then passed to the DataPower XG45 appliance. The appliance performs the actual authentication against a user registry, and if successful, returns an authentication success message back to the Worklight server. When authentication is successful, the main screen of the application is displayed and the user can proceed.

Integrating the XG45 with the appliance

The application changes required to integrate the XG45 appliance are included in a new file called `Login.js`. To create this new file and add the application changes, use the following steps:

1. Expand the `CompanyAirlines` project in Worklight Studio.
2. Expand the `apps/common/js` folder of the `LuggageTrackerDojo` application.
3. Create a new file by right-clicking the `js` folder and selecting **New** → **JavaScript Source File**. Enter the name `Login.js` in the dialog box and then click **Finish**.
4. Add the `dojo/on` and `dijit/registry` modules to handle mobile application events by adding the following lines to the file:


```
define(["dojo/on", "dijit/registry"], function(on, registry){
    });
```
5. When the user clicks the Login button, the credentials are passed to the DataPower appliance using the `authDP` method. To enable this to occur, add the `loginButtonClicked` method, shown in Example 5-28, to the `Login.js` file, placing it within the `define` block that was just added in step 4.

Example 5-28 JavaScript source for `loginButtonClicked` function

```
function loginButtonClicked(){
    var username = registry.byId("username").get("value");
    var password = registry.byId("password").get("value");
    if (!username){
        alert("User name cannot be empty.");
        return;
    }
    authDP(username, password);
}
```

6. The call to the XG45 appliance is implemented with the authDP function. To create this function, add the lines in Example 5-29 to the Login.js file, placing them above the loginButtonClicked function that you added in step 5 on page 282 (directly after the line that starts with define([]).

Example 5-29 Function to offload authentication to the XG45 appliance

```
function authDP(username, password){
    $.ajax("http://9.186.9.99:7878/authenticateUser", {
        success: function(){
            var invocationData = {
                adapter : "LuggageTrackerAdapter",
                procedure : "setUserIdentity",
                parameters : [ username, password ]
            };

            loginChallengeHandler.submitAdapterAuthentication(invocationData, {});
            registry.byId("loginView").performTransition("luggageInputView", 1,
"slide");
        },
        error: function(xhr, status, error){
            alert("Failed: " + status + " ; " + error);
        },
        headers: {
            "Authorization": "Basic " + Base64.encode(username+":"+password)
        }
    });
}
```

If authentication is successful, the LuggageTrackerAdapter (created in 4.7.2, “Creating the authentication adapter” on page 91) is called to set the user identity in IBM Worklight Server.

This function also uses the webtoolkit JavaScript base64 library that was added to the development environment in “Adding a base 64 encoding package” on page 218. The library handles the encoding and decoding of the authorization header into the base 64 format that is expected by the XG45 appliance.

7. Finally, add the connection between the button being clicked and the loginButtonClicked function, shown in Example 5-30, to the Login.js file. Place this code near the end of the file, inside the define block, just before the final line that contains the following characters: "});

Example 5-30 Binding the login button to the loginButtonClicked function

```
return {
    initView: function(){
        on(registry.byId("loginBtn"), "click", loginButtonClicked);
    }
};
```

8. Save and close the Login.js file.

The contents of the Login.js file are in “Dojo mobile application source files” on page 360.

The DataPower XG45 appliance authentication is now integrated with the application.

The final process is to ensure that the base64 library is loaded in the LuggageTrackerDojo.html file. Use the following steps:

1. Open the LuggageTrackerDojo.html file located in the common folder of the LuggageTrackerDojo application.
2. Switch to the Source tab.
3. Locate the line containing the <script> tag that references src="dojo/dojo.js" and add the following line immediately below it:

```
script src="js/base64.js"></script>
```
4. Save and close the LuggageTrackerDojo.html file.

5.4.5 Building the new mobile application

Building the MyLuggage mobile application follows a process that is similar to the one used to build the LuggageTracker application. You must complete these steps to get the application ready to be deployed into the proper environment (for example, quality assurance, pre-production, or production).

To build the final application the following files must be generated:

- ▶ Worklight adapters (multiple .adapter files within AirlineAdapterProject)
- ▶ Worklight web application (.war file)
- ▶ Worklight mobile application (multiple .wlap files)
- ▶ Native Android application (.apk file)

Before you begin, confirm that you imported all projects that were previously created in your Worklight Studio workspace. The instructions presented in this book put all of these projects in the same workspace, but in a real-life development effort, the components are created by various developers in various roles throughout the IT organization.

The following projects should be present in your Worklight Studio workspace:

- ▶ CompanyAirlines (the user interface)
- ▶ AirlineAdapterProject (the adapters)

Within an enterprise multiple target environments exist, such as development, quality assurance, pre-production, and production. For this book, development and pre-production are the only target environments. The development environment for Company Airline A's mobile development team is Worklight Studio. The pre-production environment is described in 5.1.4, "Expanding the topology" on page 212.

Several tasks must be completed to build the mobile application for a specific target environment:

- ▶ Verifying the servers and ports
- ▶ Setting the application context root
- ▶ Building the adapters
- ▶ Building the application and mobile application files
- ▶ Exporting the Android application

Verifying servers and ports

Before you can start bringing together the application components, you must verify that all server addresses match the environment where the mobile application will be deployed. For example, the pre-production environment normally replicates the production environment, although the server names probably differ.

Use the following steps to edit the necessary files to ensure that the server names and port numbers match targeted deployment environment:

1. Open the `MapAdapter.xml` file in the `adapters/MapAdapter` folder of the `AirlineAdapterProject` project, as shown in Figure 5-52.

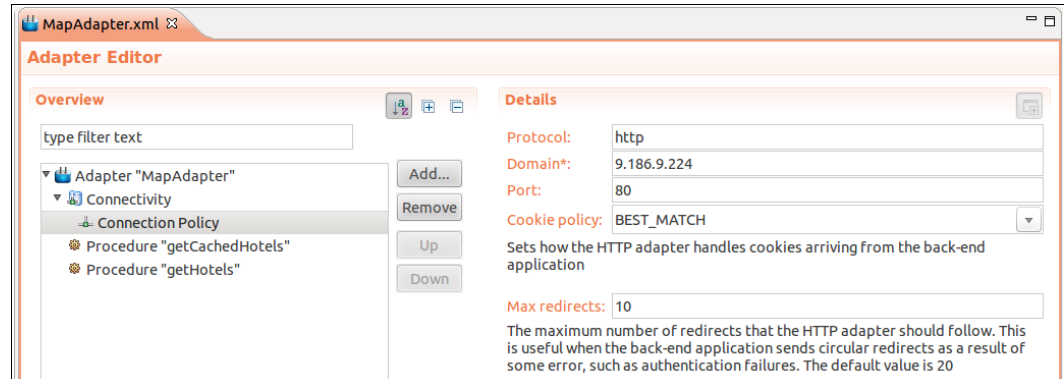


Figure 5-52 Verifying the connection information for the adapter

2. Verify that the Connection Policy details in the file (protocol, domain, and port) match the details for the back-end services provider in the target environment.
3. Save and close the `MapAdapter.xml` file.
4. Open the `BusinessServicesAdapter.xml` file in the `adapters/BusinessServicesAdapter` folder of the `AirlineAdapterProject` project.
5. Verify that the Connection Policy details in the file (protocol, domain, and port) match the details for the back-end services provider in the target environment.
6. Save and close the `BusinessServicesAdapter.xml` file.
7. Open the `LuggageTrackerAdapter.xml` file in the `adapters/LuggageTrackerAdapter` folder of the `AirlineAdapterProject` project.
8. Verify that the Connection Policy details in the file (protocol, domain, and port) match the details for the back-end services provider in the target environment.
9. Save and close the `LuggageTrackerAdapter.xml` file.

Setting the application context root

Next, the application descriptor file must be modified to use the host name or IP address of the Worklight Server. In most cases, the development team will use the local development server provided by Worklight Studio; the pre-production and production environments will have their own Worklight Server. To ensure that the mobile application is configured with the correct Worklight Server URL, use the following steps:

1. Open the `application-descriptor.xml` file in the `apps/LuggageTrackerDojo` folder of the `CompanyAirlines` project.
2. Switch to the Source tab.
3. Locate the line that contains the `<worklightServerRootURL>` tag.

4. Change the value within the tag to match the target environment using the following format, where `<wl_server_name>` is the server name of your IBM Worklight Server:

`http://<wl_server_name>:9080/luggageTracker`

Both mobile applications (LuggageTracker and MyLuggage) have the same context root, `luggageTracker`.

5. Save and close the `application-descriptor.xml` file.

Building the adapters

The MyLuggage application uses adapters to connect the mobile application to the company's back-end business services, to obtain the hotel information, and to implement the server-side authentication model. The following adapters were created and tested by a mobile server developer in 5.3, "Changes to the back-end services" on page 232:

- ▶ `BusinessServicesAdapter`
- ▶ `MapAdapter`
- ▶ `LuggageTrackerAdapter`

Now you must build the adapters to generate the files that will be used to deploy them to Worklight Server. Use the following steps:

1. Expand the `AirlineAdapterProject` project and then expand the adapters folder.
2. Right-click the `BusinessServicesAdapter` folder and select **Run As** → **Deploy Worklight Adapter**. This builds the adapters to generate the deployable adapter file, `BusinessServicesAdapter.adapter`. It also deploys this adapter to the built-in Worklight Server within Worklight Studio.
3. Repeat step 2 for `MapAdapter` in the `MapAdapter` folder.
4. Repeat step 2 for `LuggageTrackerAdapter` in the `LuggageTrackerAdapter` folder.

The three adapter files (`BusinessServicesAdapter.adapter`, `MapAdapter.adapter`, and `LuggageTrackerAdapter.adapter`) are now located in the `bin` folder of `AirlineAdapterProject` and will be used to deploy the adapters to Worklight Server in the target environment.

The `MapAdapter` also requires three additional JAR files:

- ▶ `MapAdapterProject-customization.jar`, located at the `bin` folder of `MapAdapter`
- ▶ WebSphere eXtreme Scale client API JAR (`ogclient.jar`), located in the `server/lib` folder of `MapAdapter`
- ▶ The JavaScript utility JAR (Mozilla Rhino in this example), located in the `server/lib` folder of `MapAdapter`

Ensure that these three files are copied to the local file system of IBM Worklight Server and then update the class path of IBM Worklight Server to include these JAR files.

Building the web application archive and mobile application files

For the MyLuggage application, the following project-specific web application archive and Worklight application files must be built so that they can be deployed to Worklight Server in the target environment:

- ▶ `LuggageTrackerDojo-android.wlapp`
- ▶ `LuggageTrackerDojo-common.wlapp`
- ▶ `CompanyAirlines.war`

Build the files the following steps:

1. Expand the CompanyAirlines project and then expand the apps folder.
2. Right-click the LuggageTrackerDojo folder and select **Run As** → **Build All and Deploy**.

The web application archive and Worklight application files are now built and are located in the bin folder of the CompanyAirlines project. The source code for the Android application has been generated, and a new project, CompanyAirlinesLuggageTrackerDojoAndroid, has been created and is now visible in Project Explorer. This project contains the source code that is required to build and export the Android binary file (the .apk file) that is described in the next subsection.

Exporting the Android application

Export the Android application, which generates and signs the .apk file:

1. Right-click the CompanyAirlinesLuggageTrackerDojoAndroid project folder and then select **Android Tools** → **Export Signed Application Package**.
2. The Export Android Application wizard is displayed. On the first page of the wizard, the Project field is pre-filled, as shown in Figure 5-53. Confirm that the listed project is the correct one and then click **Next**. If it is not the correct project, click **Browse** to find the correct one.

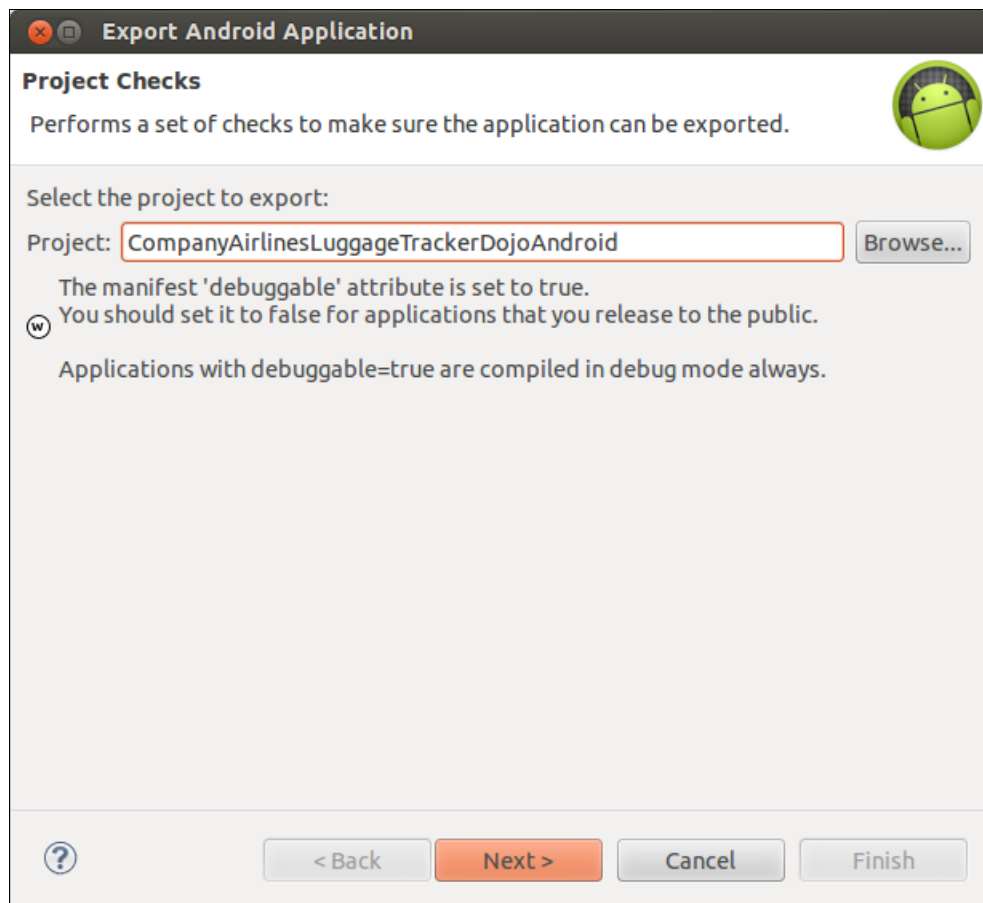


Figure 5-53 Pre-filled project field in the Export Android Application dialog

3. The Keystore selection page of the wizard (Figure 5-54) is displayed so you can provide a keystore (new or existing) for the application signing certificate. This certificate is used to sign the Android application (.apk file).

The same keystore is used for both the LuggageTracker and MyLuggage applications, so select **Use existing keystore**. Next, click **Browse** to select the keystore from the keystore folder of the CompanyAirlines project, providing the same password used for LuggageTracker, which was password.

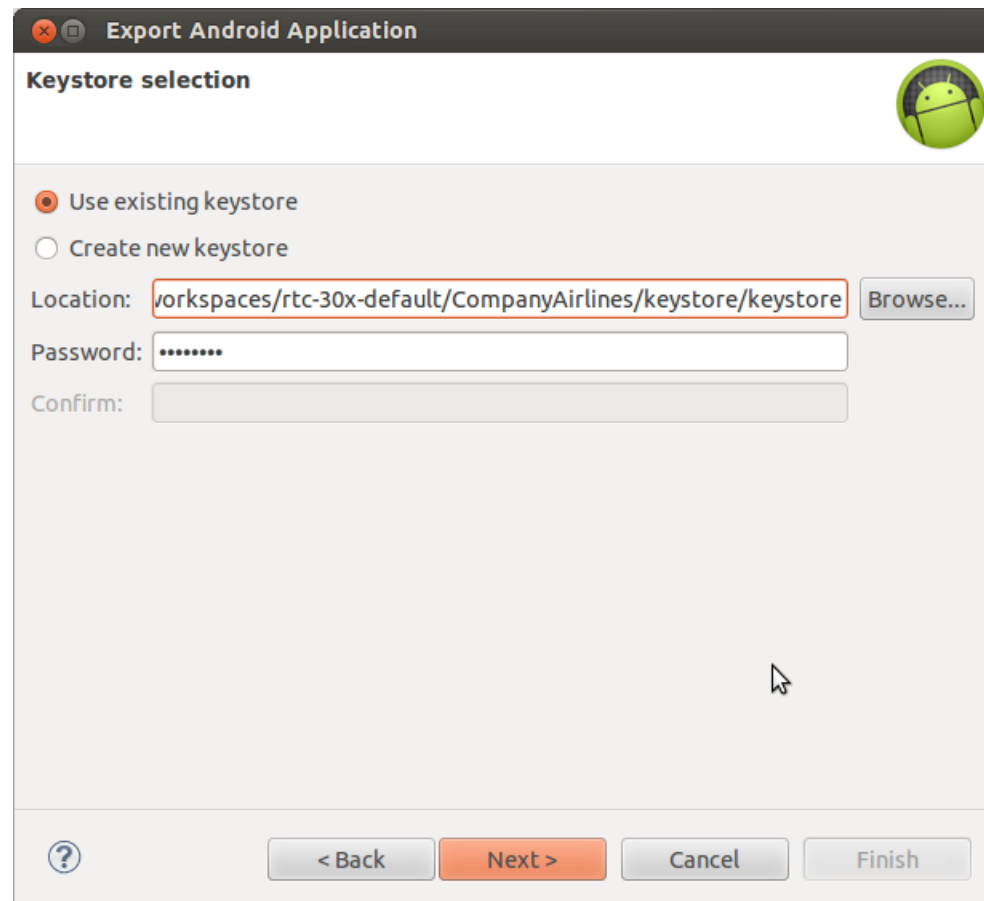


Figure 5-54 Selecting an existing keystore

4. The Key alias selection page of the wizard is displayed (Figure 5-55) so you can select a key. Select **Use existing key** and then choose the key alias that you created for LuggageTracker and provide the needed password (password in this example). Then, click **Next**.

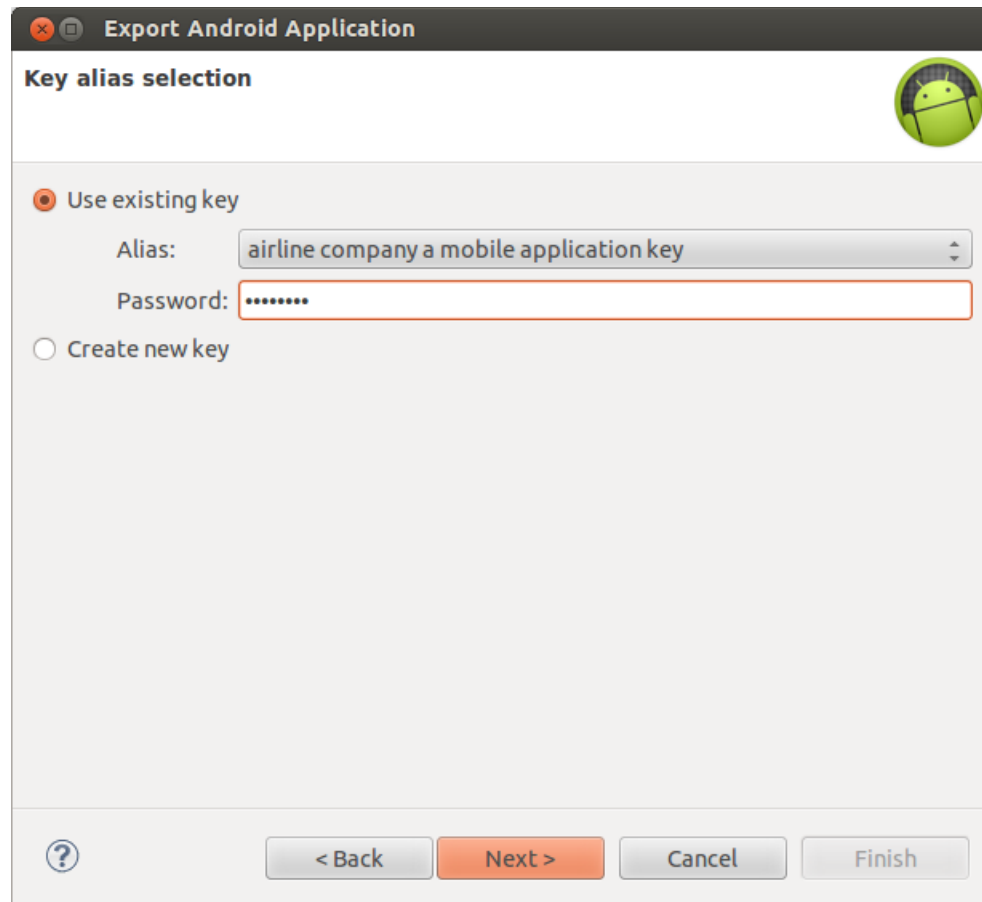


Figure 5-55 Selecting the existing key

5. The final page of the wizard, Destination and key/certificate checks, is now displayed. In the Destination APK file field, designate a location to store the signed Android application. For purposes of this book, click **Browse** and then, from the bin folder within the CompanyAirlines project, select **CompanyAirlinesLuggageTrackerDojoAndroid.apk** as shown Figure 5-56.

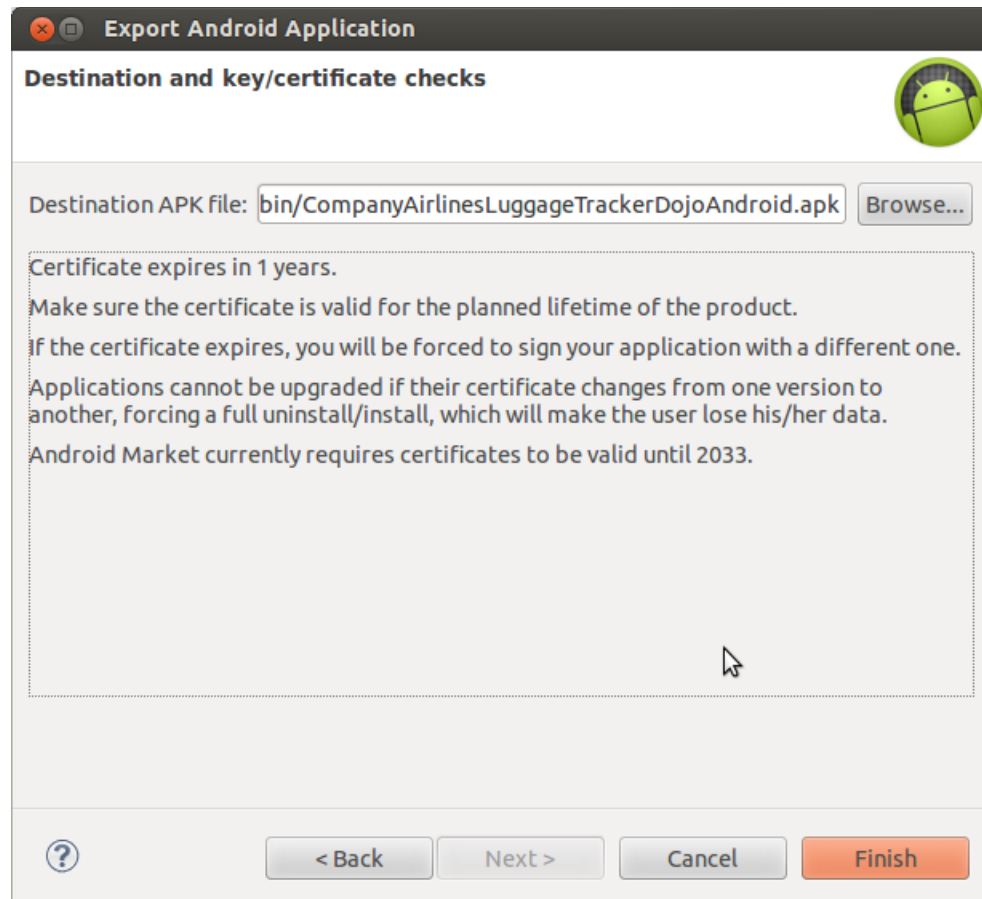


Figure 5-56 Selecting the destination folder for the signed Android application

6. Click **Finish**.

The signed Android mobile application is now in the bin folder.

5.5 Unit testing

Unit testing of MyLuggage can be done using the same process as was used for LuggageTracker (see 4.8.5, “Unit testing the application” on page 150).

5.6 Deploying the application

In contrast to the LuggageTracker application, MyLuggage will be used by passengers of the airline, not employees. Most of the tasks involved in deploying the new application are the same as for LuggageTracker, although some specifics differ slightly.

The system administrator is still responsible for deploying both the mobile application and the needed adapters to Worklight Server, and for updating the application when new versions become available. Airline passengers will install the MyLuggage from their platform vendor's application stores instead of from the airline's internal application center, which is used only for distributing applications to employees. The system administrator is responsible for publishing the application to the vendor-specific stores. For the Android platform, this is Google Play.

The high-level steps that must be completed to deploy the application are as follows:

- ▶ Deploying the adapters
- ▶ Deploying the project-specific web application archive
- ▶ Deploying and updating the Worklight application
- ▶ Deploying the application to the vendor-specific application store

Deploying the adapters

MyLuggage has three adapters that must be deployed for the application to operate:

- ▶ MapAdapter
- ▶ BusinessServicesAdapter
- ▶ LuggageTrackerAdapter

These adapters are deployed to Worklight Server by the system administrator using the steps in 4.9.4, “Deploying the Worklight adapters” on page 178. Because the MyLuggage application has new business services compared to LuggageTracker, the system administrator must ensure that these new business services are available prior to deploying the adapters to Worklight Server.

Deploying the project-specific web application archive

MyLuggage uses the same web application archive (CompanyAirlines.war) as LuggageTracker. Deploy the updated web application archive using the steps in 4.9.2, “Deploying the project-specific web application archive” on page 174.

Deploying and updating the Worklight application

The MyLuggage application has two Worklight application files (one containing the common parts and another containing the Android-specific parts):

- ▶ LuggageTrackerDojo-common.wlapp
- ▶ LuggageTrackerDojo-android-1.0.wlapp

The Worklight applications must be deployed on the Worklight server to make the mobile solution fully functional. The Worklight application files contain the web artifacts associated with the hybrid application and enables the artifacts to be updated without requiring the entire mobile application to be reinstalled.

For MyLuggage, deploy the two application files using the steps in 4.9.3, “Deploying the Worklight application” on page 176.

Deploying the application to the vendor-specific application store

MyLuggage will be used by customers of the airline who will install the application from a vendor specific application store.

The system administrator is responsible for publishing the application in the vendor application store. Each application store has its own procedures and guidelines dictated by the vendors. For example, some stores require a review before the application is made available to users to download, so this potential delay must be accounted for within the process.

Because each vendor application store publishing process differs, the details for publishing the application in the application store is not covered here. Details about publishing applications to Google Play are available from Google:

<https://play.google.com/apps/publish>



Installation and configuration

This chapter lists the specific installation and configuration steps completed by the authors in developing the sample platforms and solutions that are depicted in this book. It is not intended to replace the formal product documentation for these products and does not cover all installation options or supported platforms.

The following topics are covered:

- ▶ 6.1, “Installing products with IBM Installation Manager” on page 294
- ▶ 6.2, “Installing IBM Worklight Server” on page 295
- ▶ 6.3, “Installing IBM Worklight Studio” on page 300
- ▶ 6.4, “Installing WebSphere Application Server Liberty Profile” on page 309
- ▶ 6.5, “Installing IBM WebSphere Cast Iron” on page 315
- ▶ 6.6, “Installing IBM WebSphere eXtreme Scale” on page 319

6.1 Installing products with IBM Installation Manager

IBM Installation Manager is an enterprise deployment tool used to install, remove, and modify IBM software products. It tracks the products you install, including selectable features and maintenance updates.

Unless noted, all products covered in this chapter were installed using IBM Installation Manager Version 1.5; the steps provided here are the initial steps for installing each product.

You might already have Installation Manager installed, because it is used to install multiple IBM products. If not, download and install it using the instructions provided at this location:

http://www-947.ibm.com/support/entry/portal/Downloads/Software/Rational/IBM_Installation_Manager

For details and system requirements, see the IBM Installation Manager Information Center for versions 1 release 5:

<http://pic.dhe.ibm.com/infocenter/install/v1r5/index.jsp>

TIP: Although some products can be installed with IBM Installation Manager Version 1.5, some require a higher version. So install the latest version available.

To install a product using IBM Installation Manager, use the following steps:

1. Start the IBM Installation Manager user interface (Figure 6-1).

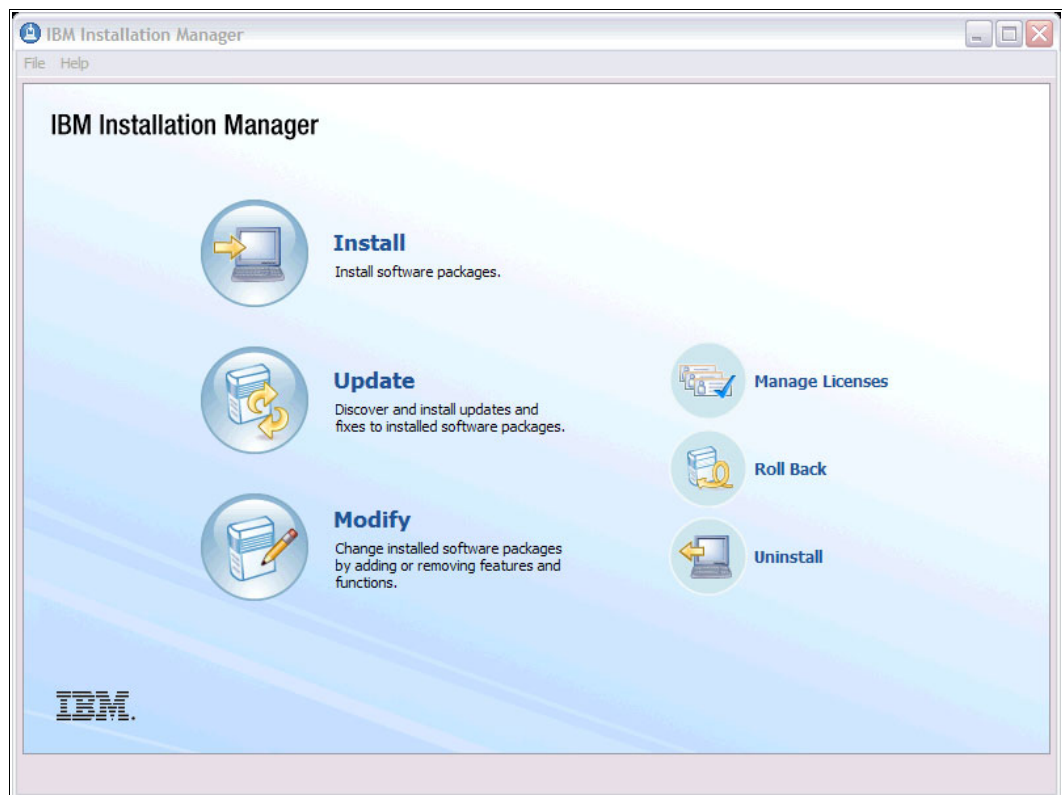


Figure 6-1 IBM Installation Manager main screen

2. Designate the location of the installation repository that contains the software packages for the product or products to be installed:
 - a. Select **File** → **Preferences** to display the Preferences window (Figure 6-2).

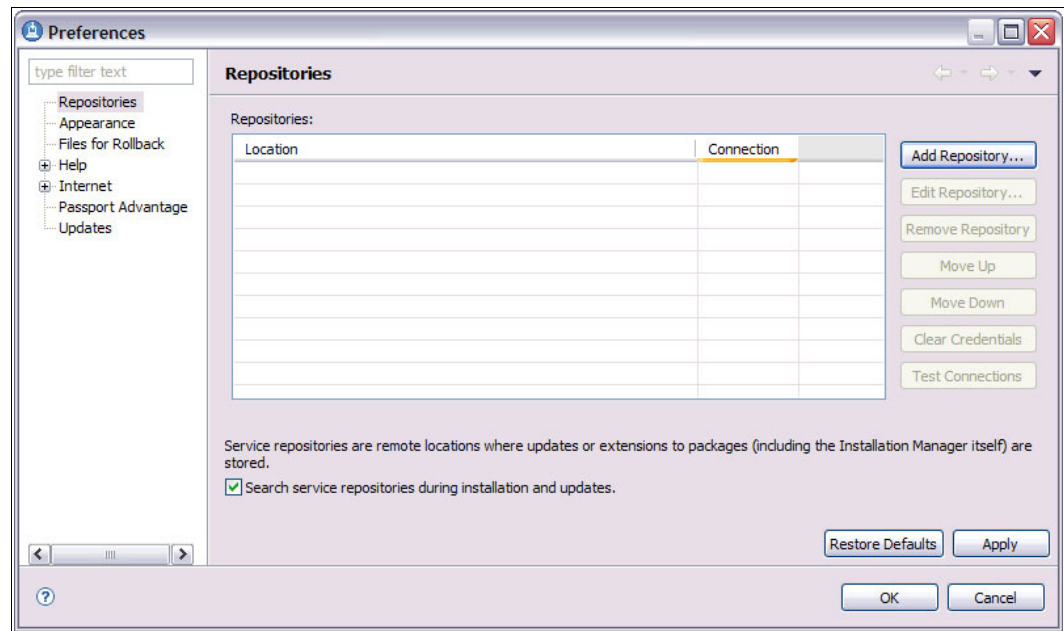


Figure 6-2 Adding a repository using the Preferences window

- b. Click **Add Repository** to add a repository location (either the physical media or the location of the downloaded product installation repository.) If you have Internet connectivity and want to install any published fixes from IBM at this time, select **Search service repositories during installation and updates**.
 - c. Click **OK** to add the repository and return to the main IBM Installation Manager window.

You are now ready to install the selected software packages. The additional steps in this process are explained in each of the package-specific sections of this chapter.

6.2 Installing IBM Worklight Server

IBM Worklight is the chosen Mobile Application Platform to support Airline Company A's mobile strategy. So, IBM Worklight Server and its components, including the Worklight Console and Worklight Application Center, must be installed.

For this book, IBM Worklight Server Enterprise Edition Version V5.0.5 was installed on a system running Red Hat Enterprise Linux Server release 6.2. The installation was performed using IBM Installation Manager Version 1.5.2, which is the minimum level required to install the product.

IBM Worklight supports multiple database management systems and multiple application servers. For simplicity, Apache Derby and IBM WebSphere Application Server V8.5 Liberty Profile, both of which are provided with IBM Worklight, were used in this scenario.

For detailed system requirements, a list of the supported operating systems, and other information about Worklight Server Enterprise Edition and its installation, go to the following address:

<http://www.ibm.com/support/docview.wss?uid=swg27024838>

6.2.1 Performing the installation

Before you begin, ensure that you have access to the IBM Worklight V5.0.5 installation repository.

To install IBM Worklight Server, use the following steps:

1. Start IBM Installation Manager and add the IBM Worklight Server installation repository as explained in 6.1, “Installing products with IBM Installation Manager” on page 294.
2. On the main IBM Installation Manager window, click **Install**.
3. Select the appropriate installation package, as shown in Figure 6-3.

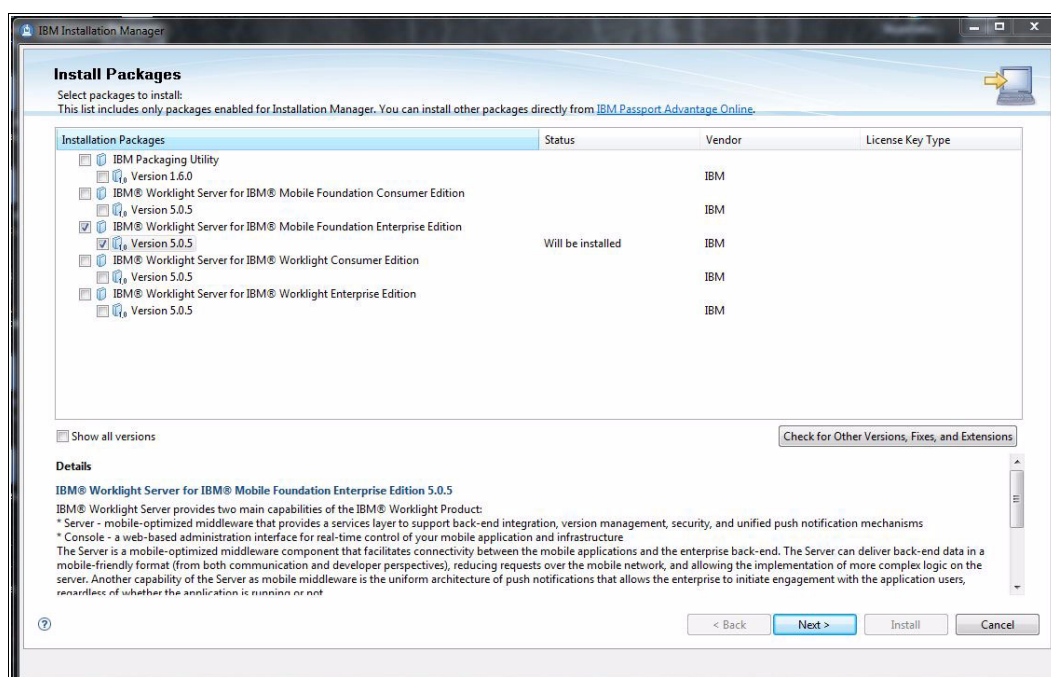


Figure 6-3 Selecting installation packages for IBM Worklight Server V5.0.5

4. Click **Next** and continue through the installation wizard, accepting the default options, until you reach the Database panel.
5. On the Database panel, define the database that IBM Worklight will use:
 - a. To match what the authors did for this book, select Apache Derby 10.8, as shown in Figure 6-4 on page 297, and then click **Next**. On the next panel, where additional database parameters can be entered, click **Next** again. The selection of Apache Derby means that no additional parameters are required and the databases are created automatically.
 - b. To use a different database, select it from the list and then click **Next**. Then, on the next panel, define the required database parameters and click **Next**.

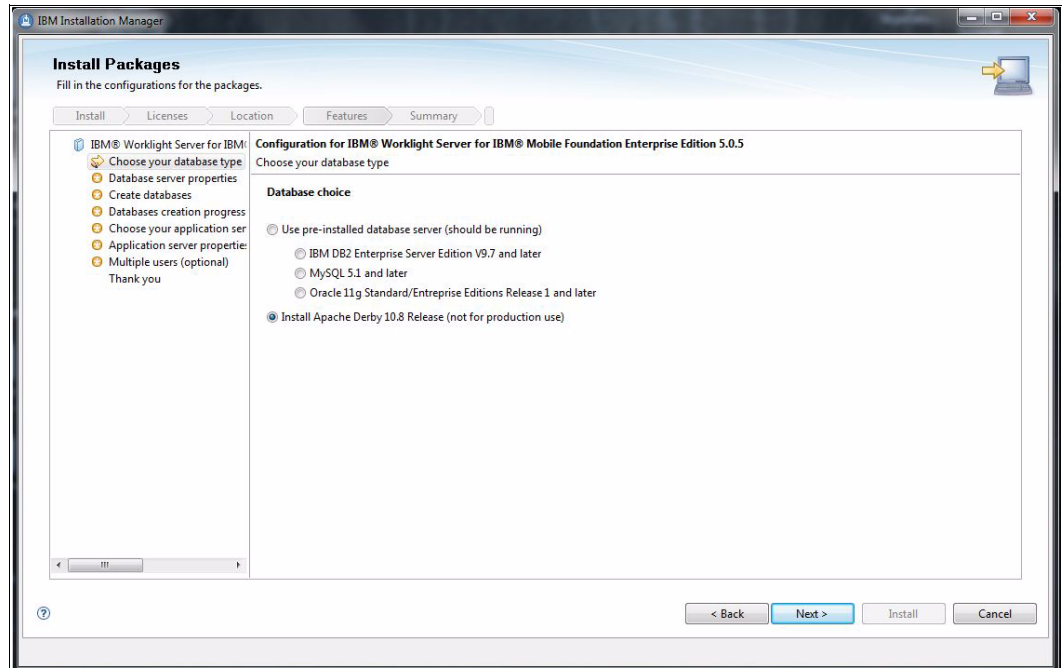


Figure 6-4 Selecting the database for IBM Worklight

6. On the Server panel, specify which application server IBM Worklight will use:
 - a. To match what the authors did for this book, select WebSphere Application Server V8.5 Liberty Profile, as shown in Figure 6-5 on page 298, and then click **Next**. On the next panel, where additional server parameters can be entered, click **Next** again. The selection of WebSphere Application Server Liberty Profile means that no additional parameters are required and the application server is created and configured automatically.
 - b. To use a different server, select it from the list and then click **Next**. Then, on the next panel, define the required server parameters and click **Next**.

Continue to follow the installation wizard, selecting options based on the designated application server, until you reach the Multiple users panel.

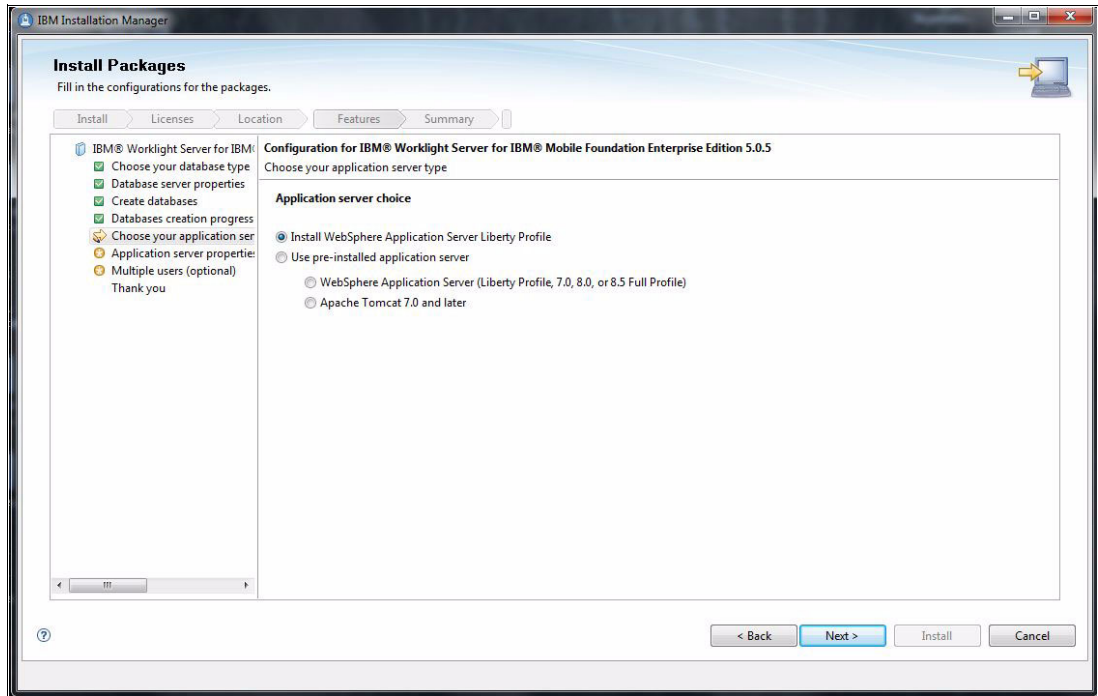


Figure 6-5 Selecting the application server

7. On the Multiple users panel, define a set of users that are permitted to stop and start the application server and access and modify the databases. Ensure that the User ID or User IDs you intend to select is in the adm group to ensure proper access rights.

For purposes of this example, select **Multiple users**, then add the user ID or user IDs to be used to start and stop the Worklight server. Then click **Next** until you get to the Install Packages summary.

8. On the Install Packages summary, review the information and then click **Install** to begin the installation.
9. When installation is complete, a confirmation window is displayed; click **Finish**.

6.2.2 Verifying the installation by starting Worklight Server

When installation is complete, you must confirm it succeeded by starting IBM Worklight Server. To determine the correct start-up procedure for your operating system, consult the *Installation* section of the IBM Worklight Server V5.0.5 Information Center:

http://pic.dhe.ibm.com/infocenter/wrklght/v5r0m5/index.jsp?topic=%2Fcom.ibm.worklight.help.doc%2Fstart%2Fr_install_wl_server.html

Figure 6-6 shows the confirmation message you receive when the Worklight Server starts successfully.

```
[root@localhost bin]# ./server start worklightServer
Server worklightServer started with process ID 19170.
[root@localhost bin]#
```

Figure 6-6 Verification that IBM Worklight Server has started

6.2.3 Verifying the installation with IBM Worklight Console

You can also verify that IBM Worklight Server successfully started by using IBM Worklight Console (Figure 6-7), which is installed with IBM Worklight Server. The console is accessed by opening a web browser and navigating to the following location (where `<server_host_name>` is the host name where IBM Worklight Server is installed):

`http://<server_host_name>:9080/worklight/console`

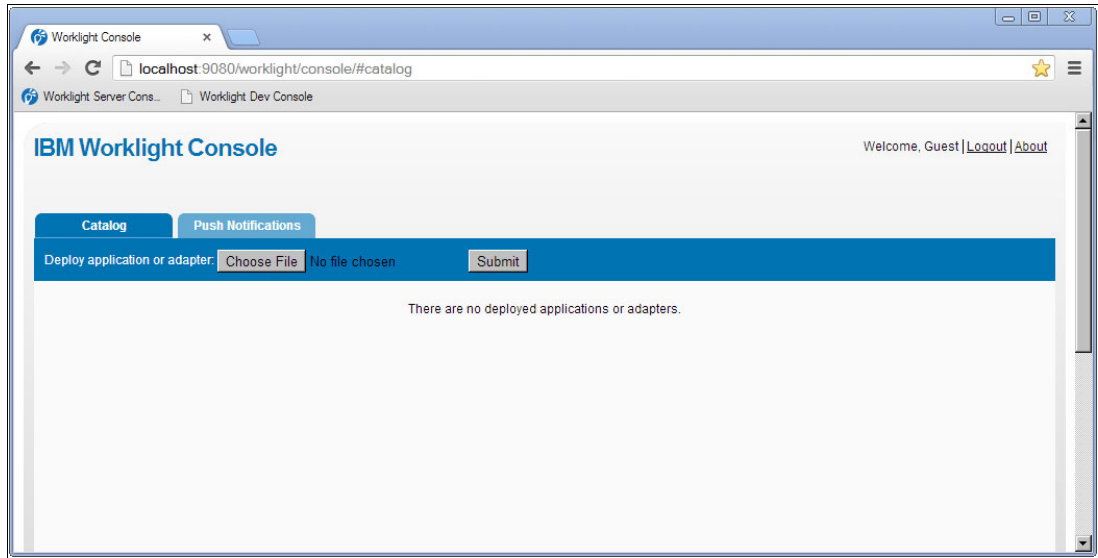


Figure 6-7 IBM Worklight Console

Troubleshooting: If you have an active firewall and notice issues with IBM Worklight Console after confirming that the server has started successfully, verify that port 9080 is open in your firewall settings.

6.2.4 Verifying the installation with IBM Application Center

You can also verify that IBM Worklight Server successfully started by using another component, IBM Worklight Application Center (Figure 6-8).

Verify the configuration of Application Center by opening a web browser and navigating to the Application Center URL, shown here (where `<server_host_name>` is the host name where IBM Worklight Server is installed):

`http://<server_host_name>:9080/applicationcenter`

Then, log in with the user ID **demo** and the password **demo**, which were defined by default during installation. If you can log in successfully, then Worklight Server was installed and configured properly.

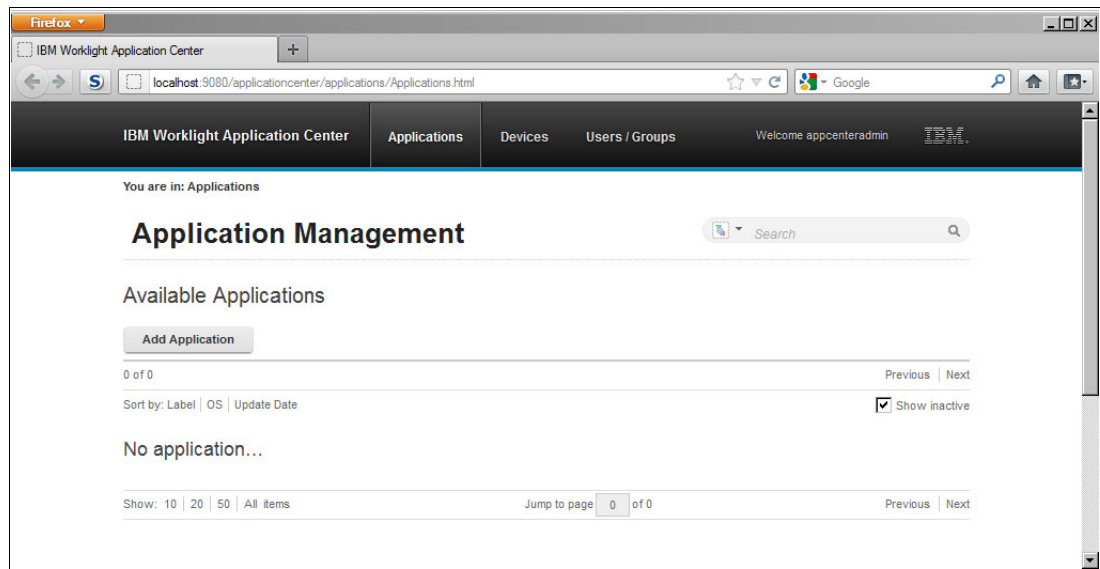


Figure 6-8 IBM Worklight Application Center

6.3 Installing IBM Worklight Studio

To support development of Airline Company A's mobile applications on multiple platforms, one instance of IBM Worklight Studio Enterprise Edition was installed on Mac OS X 10.8.2 and another instance was installed on Windows XP. For the instance on Mac OS X, a test application server using IBM WebSphere Application Server V8.5 Liberty Profile was also installed, as were the Android SDK tools for Eclipse, which are used to assist with native code development.

For detailed system requirements, a list of supported operating systems, and other information about installing IBM Worklight Studio, see the system requirements for IBM Worklight:

<http://www.ibm.com/support/docview.wss?uid=swg27024838>

6.3.1 Installing IBM Worklight Studio on Mac OS X

To install IBM Worklight Studio on Mac OS X, you must first install (or have an existing installation of) the Eclipse IDE for Java EE Developers Version 3.6.2 or later, because the process uses the Install New Software function of the IDE.

These instructions install and configure IBM Worklight Studio V5.0.5 on Mac OS X 10.8.2 using Eclipse IDE for Java EE Developers Version 3.7.2, all 64-bit. Before beginning, ensure that you have access to the IBM Worklight Studio Update Site Archive, named `iws_eclipse_5.0.5.zip`.

To install IBM Worklight Studio, use the following steps:

1. Start the Eclipse IDE and click **Help** → **Install New Software**, which opens the Available Software window shown in Figure 6-9.

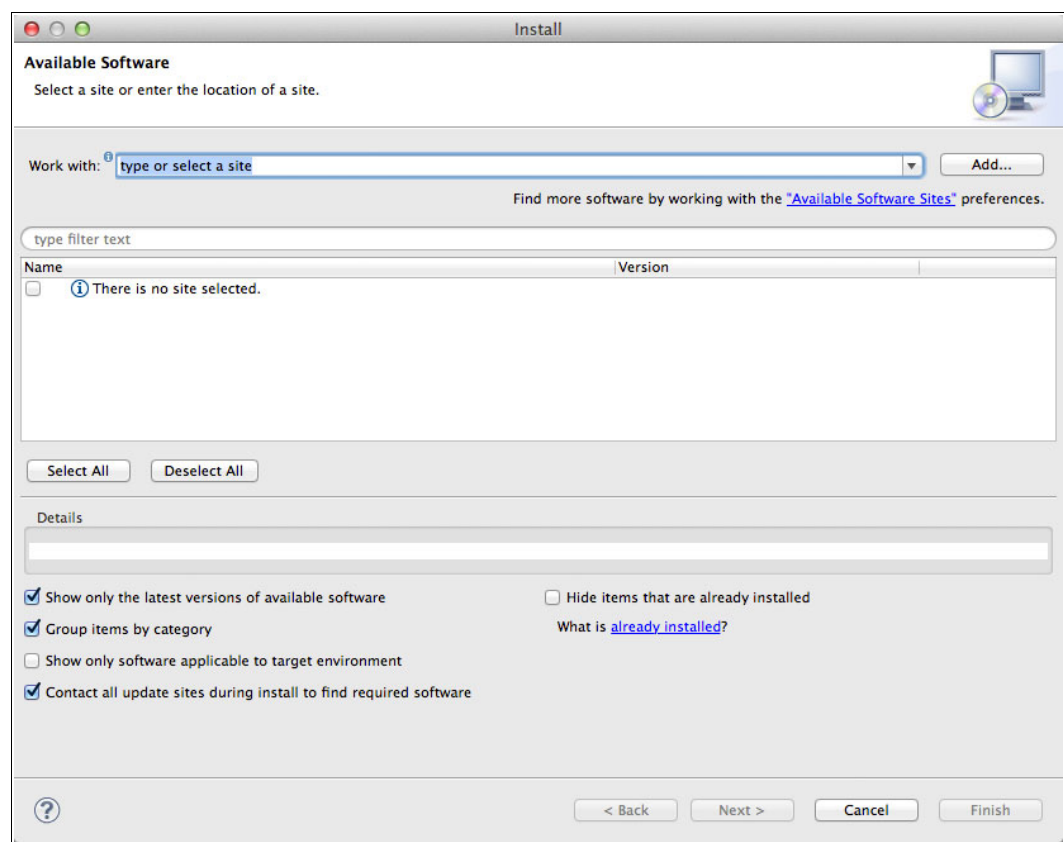


Figure 6-9 Available Software window showing no installed repositories (sites)

2. Click **Add** to display the Add Repository dialog (Figure 6-10 on page 302). Enter a repository name, such as `WL Studio 5.0.5`, and then click **Archive** to open the repository archive dialog.
3. In the repository archive dialog, find and select the archive that will be used for the installation, in this case, the IBM Worklight Studio Update Site Archive (`iws_eclipse_5.0.5.zip`), and then click **Open**. The name and archive location is now displayed in the Add Repository dialog, as also shown in Figure 6-10 on page 302.

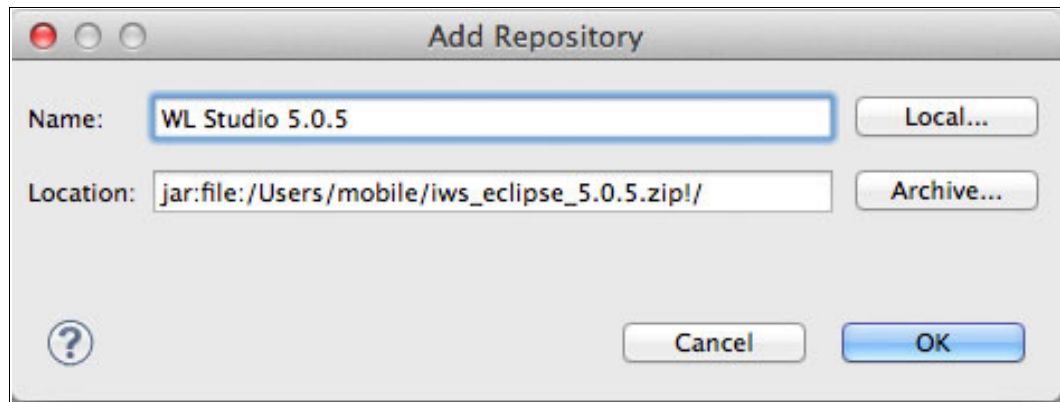


Figure 6-10 Adding the repository for Worklight Studio V5.0.5

4. Click **OK** to add the repository.
5. The features of IBM Worklight Studio V5.0.5 are now displayed for you to review in the Available Software window, as shown in Figure 6-11. For purposes of this installation, select all of the features and click **Next**.

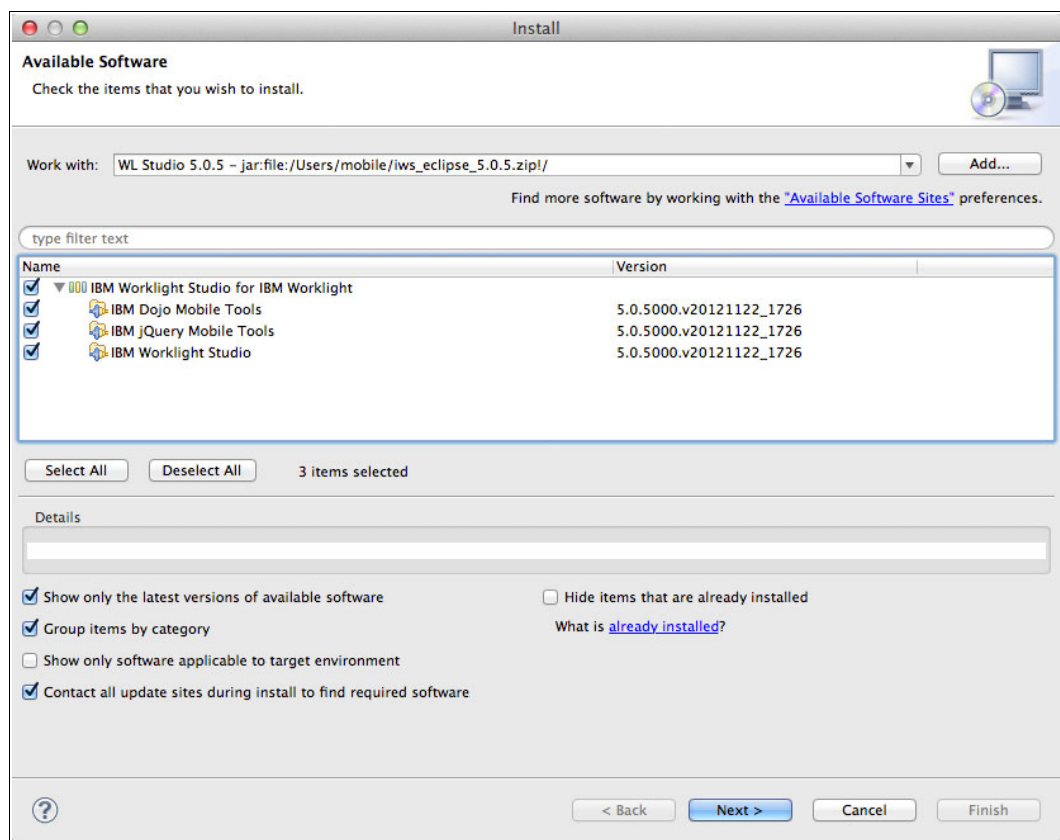


Figure 6-11 Available Software window after adding IBM Worklight Studio V5.0.5 repository

6. Eclipse calculates the dependencies for your selections. In this example, there are no additional dependent features to install, so you can click **Next**. (However, if the dependency calculation reveals that additional features are needed, you select them from the updated list that is presented here and then click **Next**.)

7. Review and accept the license agreements and click **Finish** to begin the installation.
8. When the installation is complete, click **Not Now** when prompted to restart Eclipse or to apply the changes now without restarting, and then close Eclipse.

IBM Worklight Studio is now installed and ready to use.

Troubleshooting: If you encounter issues starting IBM Worklight Studio when installed into an existing Eclipse installation, you might need to update the `eclipse.ini` file based on the information in the following IBM Worklight V5.0.5 Information Center document:

http://pic.dhe.ibm.com/infocenter/wrklight/v5r0m5/index.jsp?topic=%2Fcom.ibm.worklight.help.doc%2Fdevenv%2Ft_next_steps.html

Configuring Worklight Studio for MacBook Pro with Retina display

If you are using Worklight Studio on a MacBook Pro with Retina display, some additional configuration is necessary to fully utilize the Retina display. As with the `eclipse.ini` file in the previous steps, this process involves editing the `Eclipse Info.plist` file, so you are advised to make a backup copy of this file before proceeding.

To configure Worklight Studio for MacBook Pro with Retina display, use the following steps:

1. Open the `Info.plist` file (in this example, it is located in the following directory:
 `/Applications/eclipse/Eclipse.app/Contents/`
2. Scroll to a point near the end of the file and add the following lines of text *immediately above* the line that contains `</dict> </plist>`:
 `<key>NSHighResolutionCapable</key>`
 `<true/>`
3. Save and close the `Info.plist` file.

Complete details about MacBook Pro with Retina display configuration are available in an Eclipse documentation comment:

https://bugs.eclipse.org/bugs/show_bug.cgi?id=382972#c4

6.3.2 Installing the Android SDK on Mac OS X

The mobile application development scenario presented in this book also requires the Android Development Tools plug-in for Eclipse and the Android SDK, which contains the API libraries and tools needed to build, test, and debug native mobile applications for Android devices.

Start by obtaining the Android SDK, which is not delivered with IBM Worklight Studio and must be downloaded from this address:

<http://developer.android.com/sdk/index.html>

Decompress the downloaded Android SDK file and copy the folder to the location where you store SDKs or other development-related files (for this example, `~/dev/` was used). You use this folder later in the installation process.

To install the Android Development Tools (ADT) plug-in for Eclipse, complete these steps:

1. Launch Eclipse.
2. Click **Help** → **Install New Software** to display the Available Software window.

3. Click **Add** to open the Add Repository dialog.
4. Enter a name, such as ADT Plugin, and then type the location of the plug-in file (in this example, <https://dl-ssl.google.com/android/eclipse/>) in the location field, as shown in Figure 6-12. Then click **OK**.

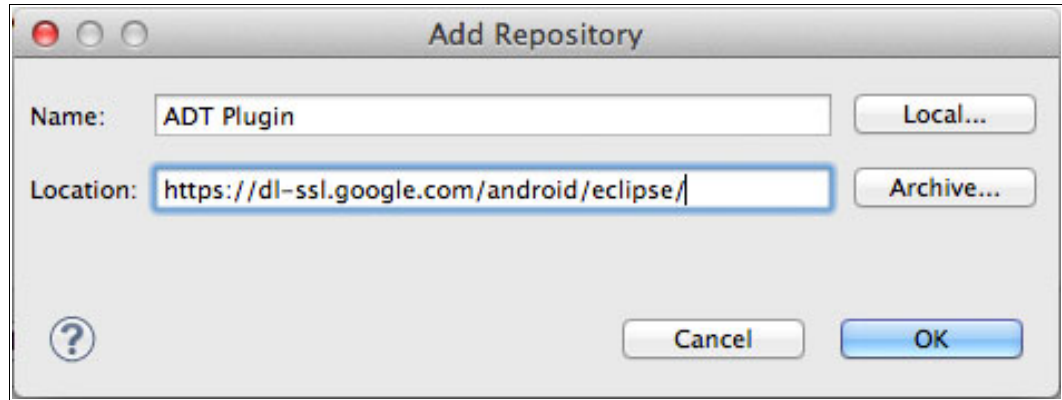


Figure 6-12 Adding the repository for the ADT Plugin

5. The Available Software window now shows the ADT plug-in features that can be installed. Select all of the Developer Tools, as shown in Figure 6-13, and then click **Next**.

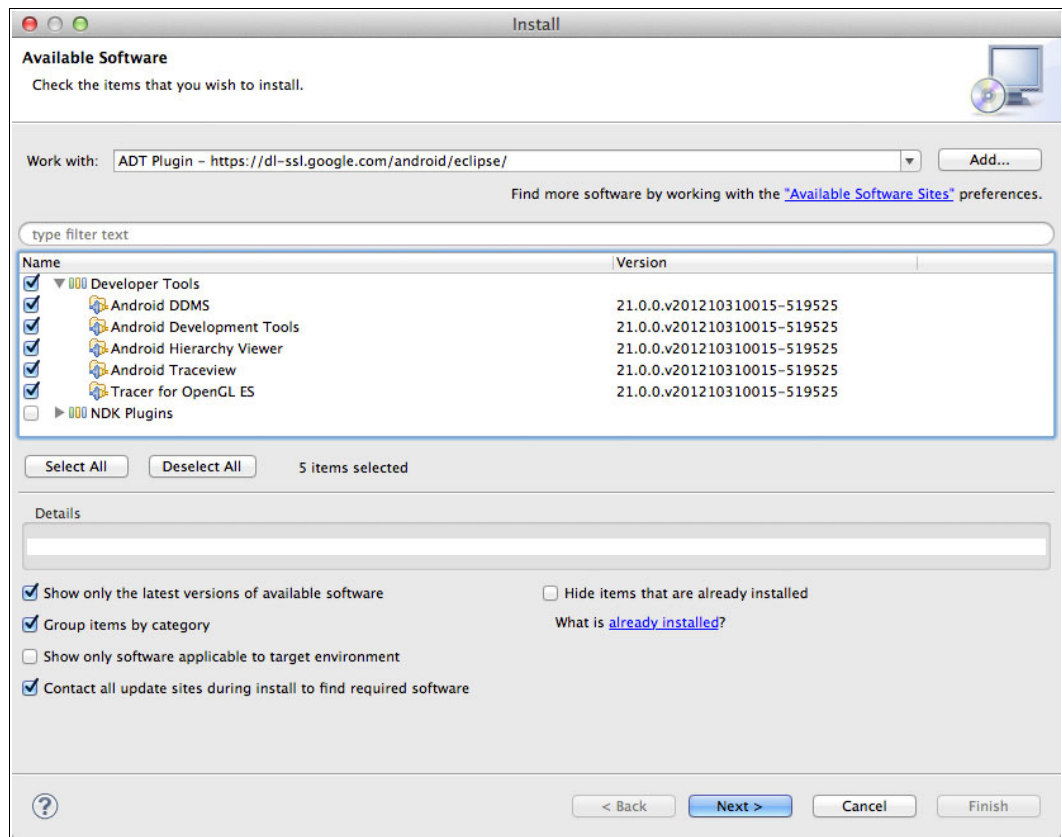


Figure 6-13 Available software for Developer Tools

6. Continue through the installation wizard, accepting the default settings, until the installation completes.
7. Restart Eclipse.
8. After Eclipse is restarted, the Welcome to Android Development window opens. In addition, you probably receive a warning that the location of the Android SDK is not set up.
If you get this warning, click **Open Preferences** on the warning message dialog to open the Preferences window (Figure 6-14) and then do the following tasks:
 - a. If you see questions about sending statistics to Google, answer as you see fit.
 - b. Enter the location of the Android SDK folder that you downloaded and decompressed, and then click **OK**.

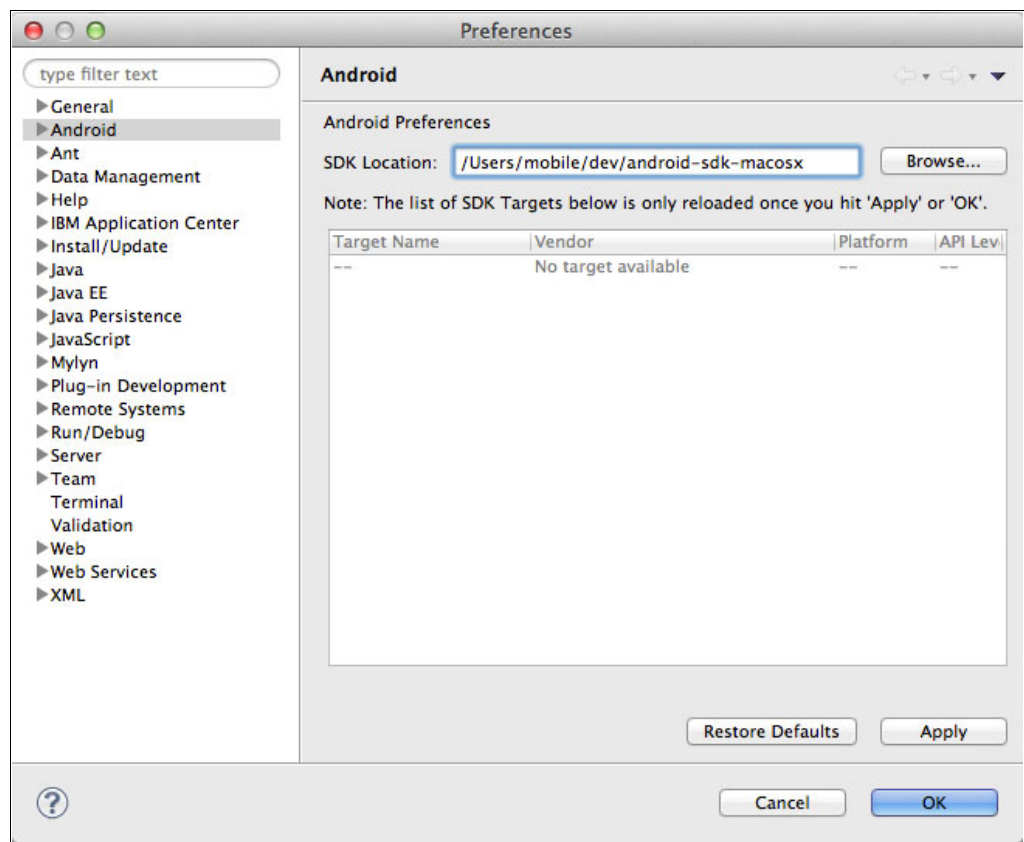


Figure 6-14 Preferences window

9. You also get a warning that the SDK Platform Tools are missing. To install them, click **Open SDK Manager** and use the following steps:
 - a. In the SDK Manager window (Figure 6-15), select **Android SDK Platform-tools**, **Android Support Library**, and the specific Android versions you will use to test your application (in this example, versions are Android 4.2 and Android 2.2).

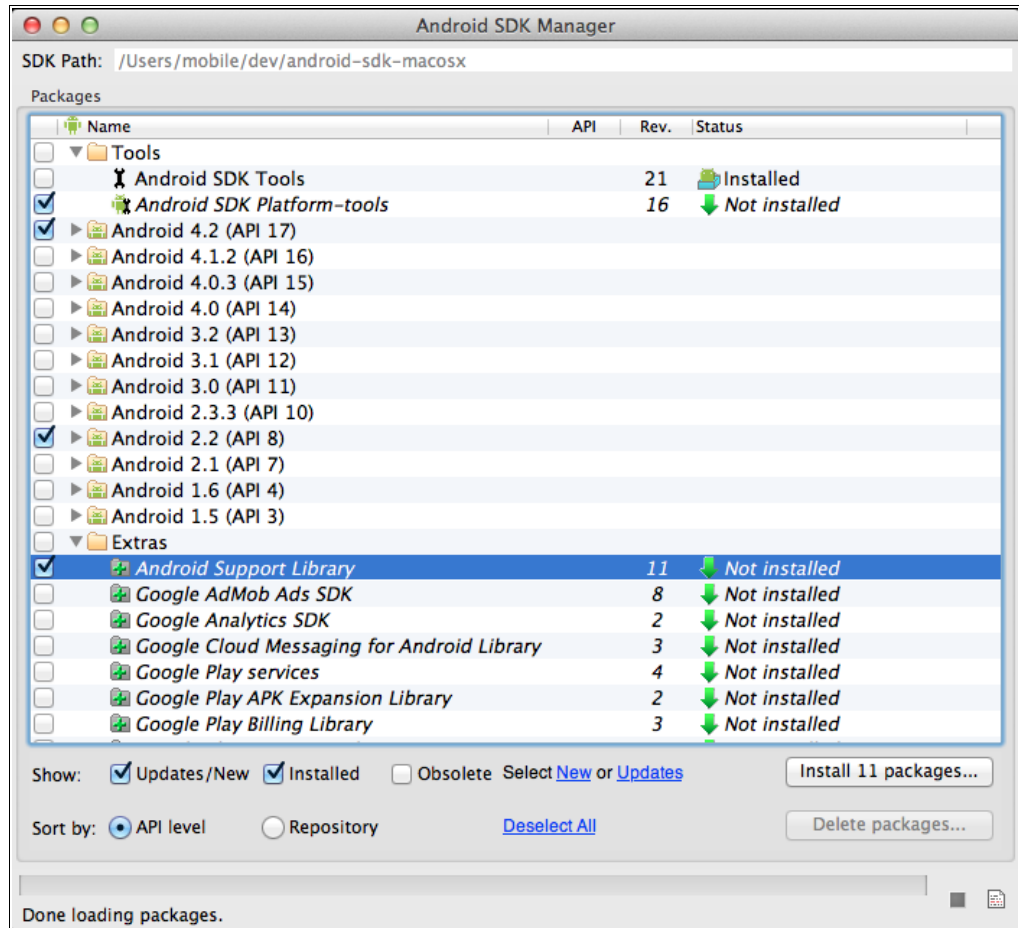


Figure 6-15 Android SDK Manager

- b. Click **Install** and continue through the steps until the installation is complete. The Android SDK Manager Log window is displayed and shows the details of the installation as it progresses.
 - c. When the installation completes, click **Close** to close the Android SDK Manager Log window.
 - d. Close the SDK Manager window.
10. In the Welcome to Android Development window, click **Cancel** because everything needed for the purposes of this book is now downloaded and installed.

Now, although the ADT plug-in and the Android SDK have been installed, you must still configure at least one virtual device to use for testing.

To configure a virtual device, use the following steps:

1. From within Eclipse, select **Android Virtual Device Manager** from the Window menu. If you do not see this option, make it visible by switching to the Java Perspective by choosing **Window** → **Open Perspective** → **Other** and then selecting **Java**.
2. In Android Virtual Device Manager, click **New** to open a dialog where you create a new Android Virtual Device (AVD).
3. In the dialog, give the AVD a name, such as Android 2.2 as shown in Figure 6-16.

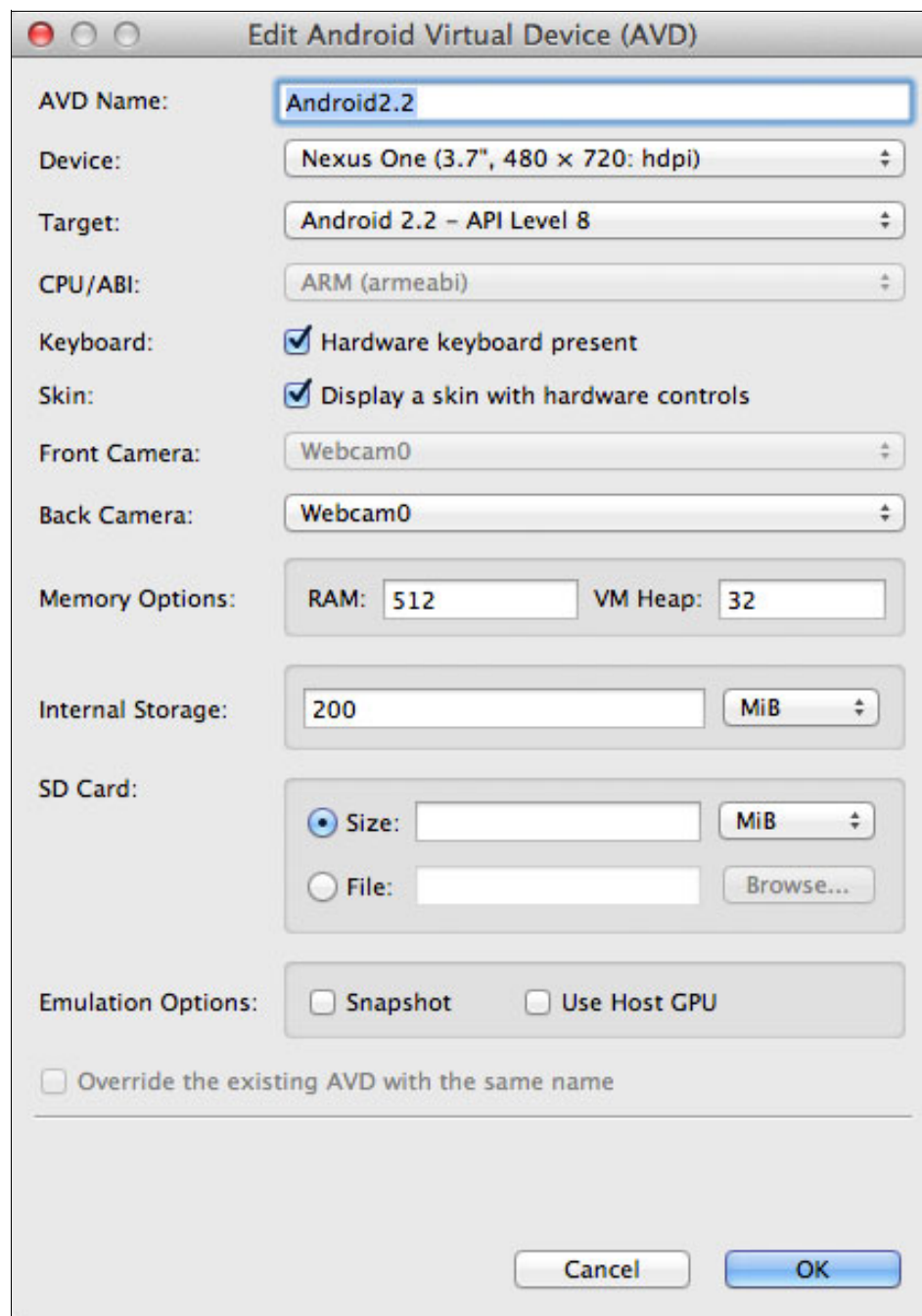


Figure 6-16 Edit Android Virtual Device dialog

4. In the Device dropdown list, select **Nexus One (3.7", 480 x 720: hdpi)**.

5. In the Target list, select **Android 2.2 - API Level 8**.
6. Click **OK**.

The new AVD is now ready for use. If additional devices are needed, repeat these steps.

6.3.3 Installing IBM Worklight Studio on Windows XP

Installing IBM Worklight Studio on Windows XP requires that Oracle Java Runtime Environment Version 1.6 or higher is available on the system. After you confirm the availability of this runtime environment, you can install IBM Worklight Studio on Windows XP as follows:

1. Start IBM Installation Manager (version 1.5.2 or later is required) and add the IBM Worklight Studio installation repository, as explained in 6.1, “Installing products with IBM Installation Manager” on page 294.
2. In the Installation Manager, click **Install**.
3. Select your IBM Worklight Studio edition and version (in this example, it is Enterprise Edition V5.0.5) and then click **Next**. Proceed through the installation wizard by accepting the default values, including acceptance of the license information, until the Feature list (Figure 6-17) is displayed.

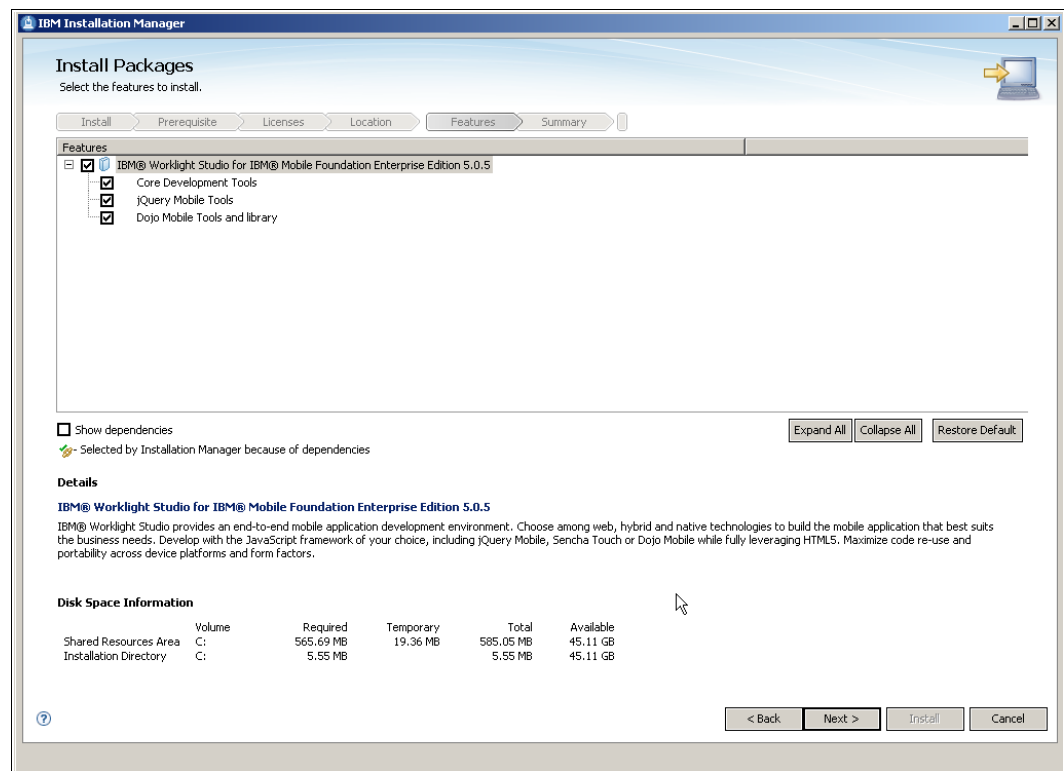


Figure 6-17 Selecting features of IBM Worklight Studio V5.0.5

4. Select all available features and then click **Next**.
5. Enter the path to the required Oracle Java runtime environment, as described at the start of this section.
6. Continue through the remainder of the installation wizard, accepting the default values, until the Summary window is displayed, and then click **Install**. When installation is completed, a confirmation message is displayed (Figure 6-18 on page 309).

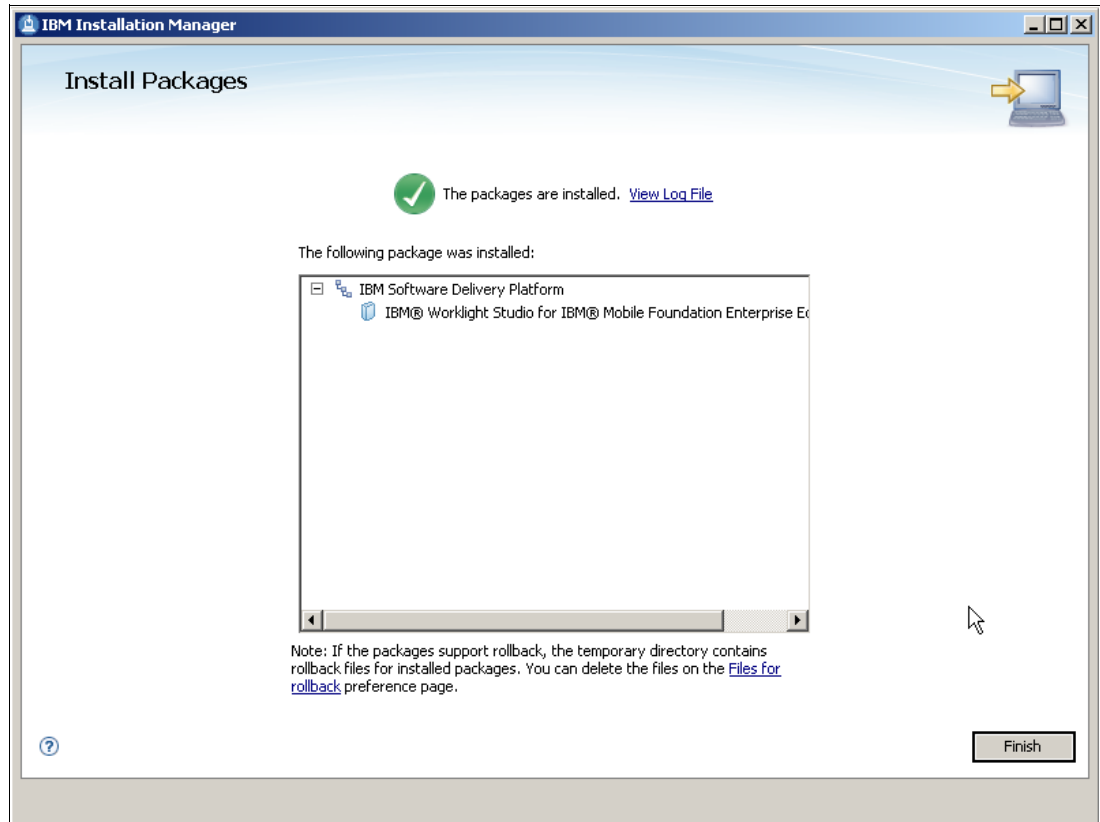


Figure 6-18 Confirmation of a successful installation

IBM Worklight Studio is now installed and ready to use on Windows XP.

6.4 Installing WebSphere Application Server Liberty Profile

IBM WebSphere Application Server 8.5 Liberty Profile was used to test the mobile applications developed for Airline Company A. These instructions explain how to install this server from the Eclipse Marketplace.

The steps are the same for both Mac OS X 10.8.2 and Windows XP, but the figures used for illustration here depict an installation on Mac OS X.

6.4.1 Installing the Liberty Profile

To install WebSphere Application Server 8.5 Liberty Profile, use the following steps:

1. Launch Eclipse.
2. Select **Help** → **Eclipse Marketplace**.

3. In the Marketplace window, search for WebSphere Liberty as shown in Figure 6-19.

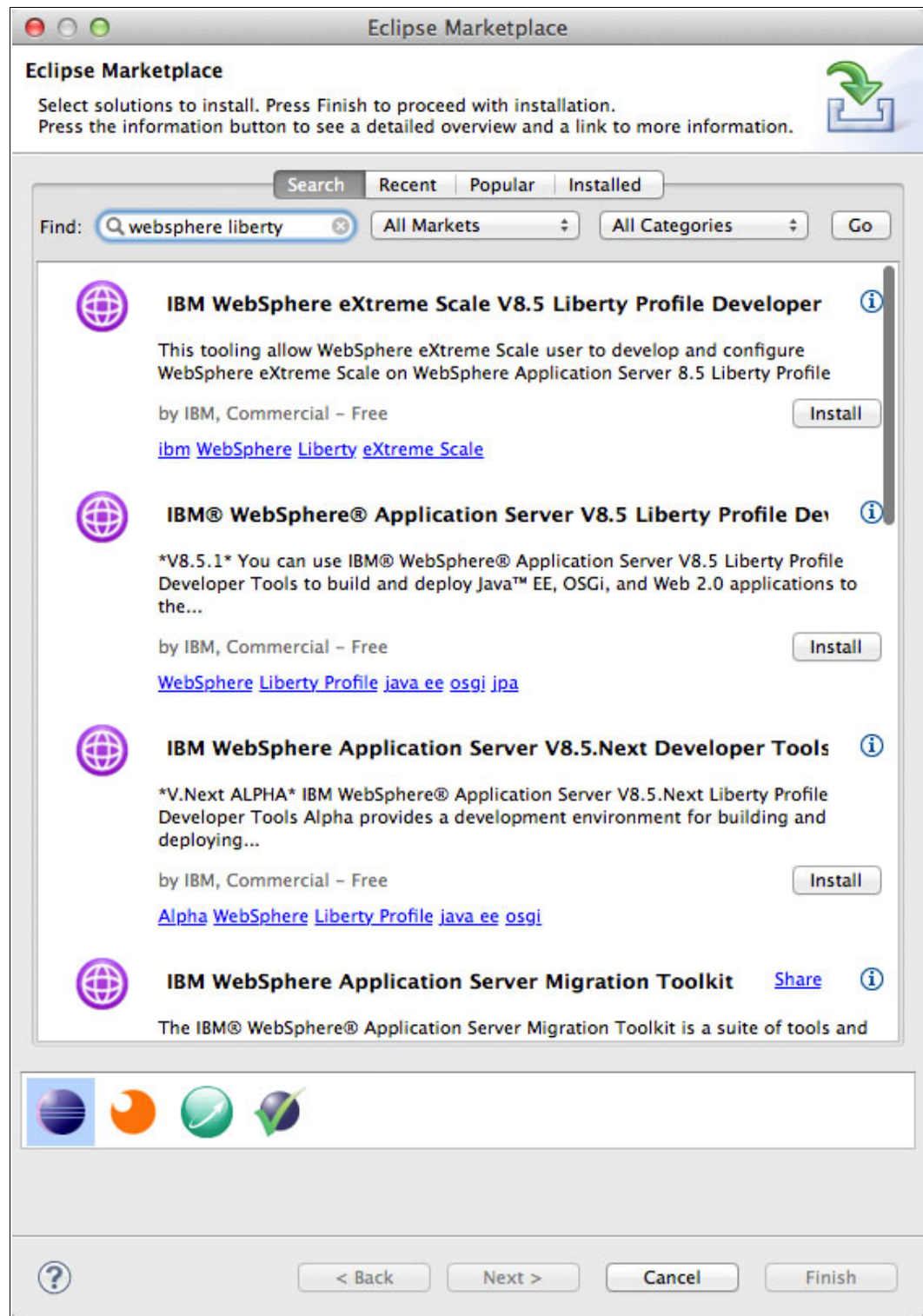


Figure 6-19 Searching for the Liberty Profile in Eclipse Marketplace

4. Find the listing for IBM WebSphere Application Server V8.5 Liberty Profile Developer Tools and click **Install**. A list of available features is displayed.

5. Select all available features and click **Next**, as shown in Figure 6-20.

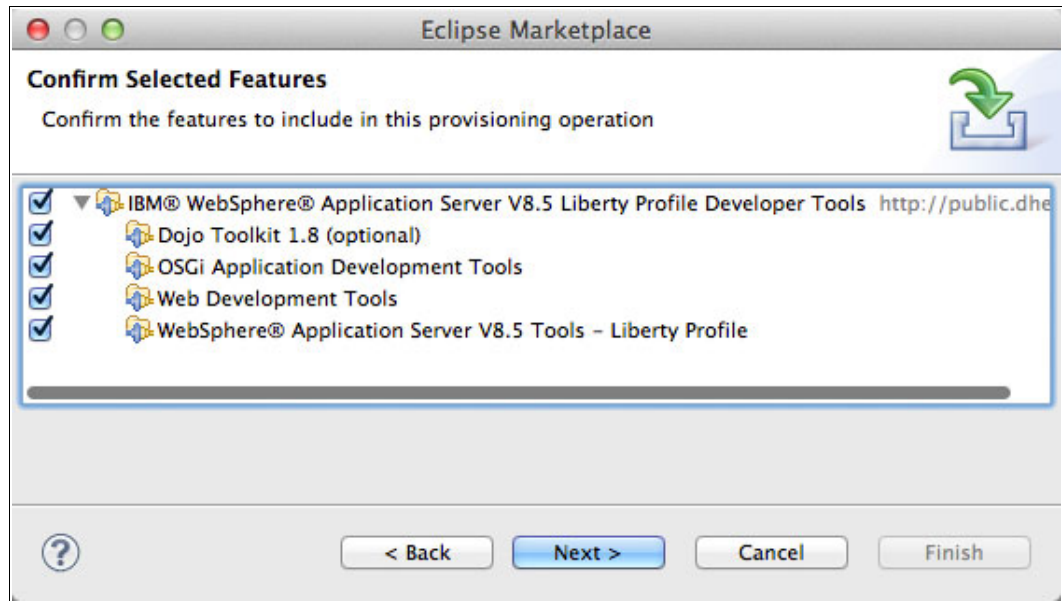


Figure 6-20 Selecting features of WebSphere Application Server V8.5 Liberty Profile Developer Tools

6. Review and accept the license agreements and then click **Finish** to start the installation.
7. When installation is complete, restart Eclipse.

6.4.2 Creating the application server instance

The next step is to create a new application server instance to be used as a test server.

1. After Eclipse is restarted, choose **Window** → **Show View** → **Servers**.
2. In the Servers tab, right-click **New** → **Server** as shown in Figure 6-21.

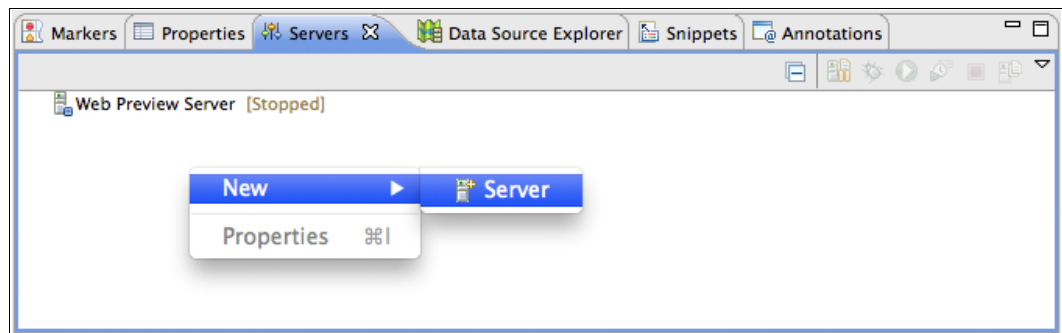


Figure 6-21 Creating a new server

3. The New Server window opens. It guides you through these steps to create a new server:
 - a. On the Define a New Server pane (Figure 6-22), select WebSphere Application Server V8.5 Liberty Profile from the list of available server types and then click **Next**.

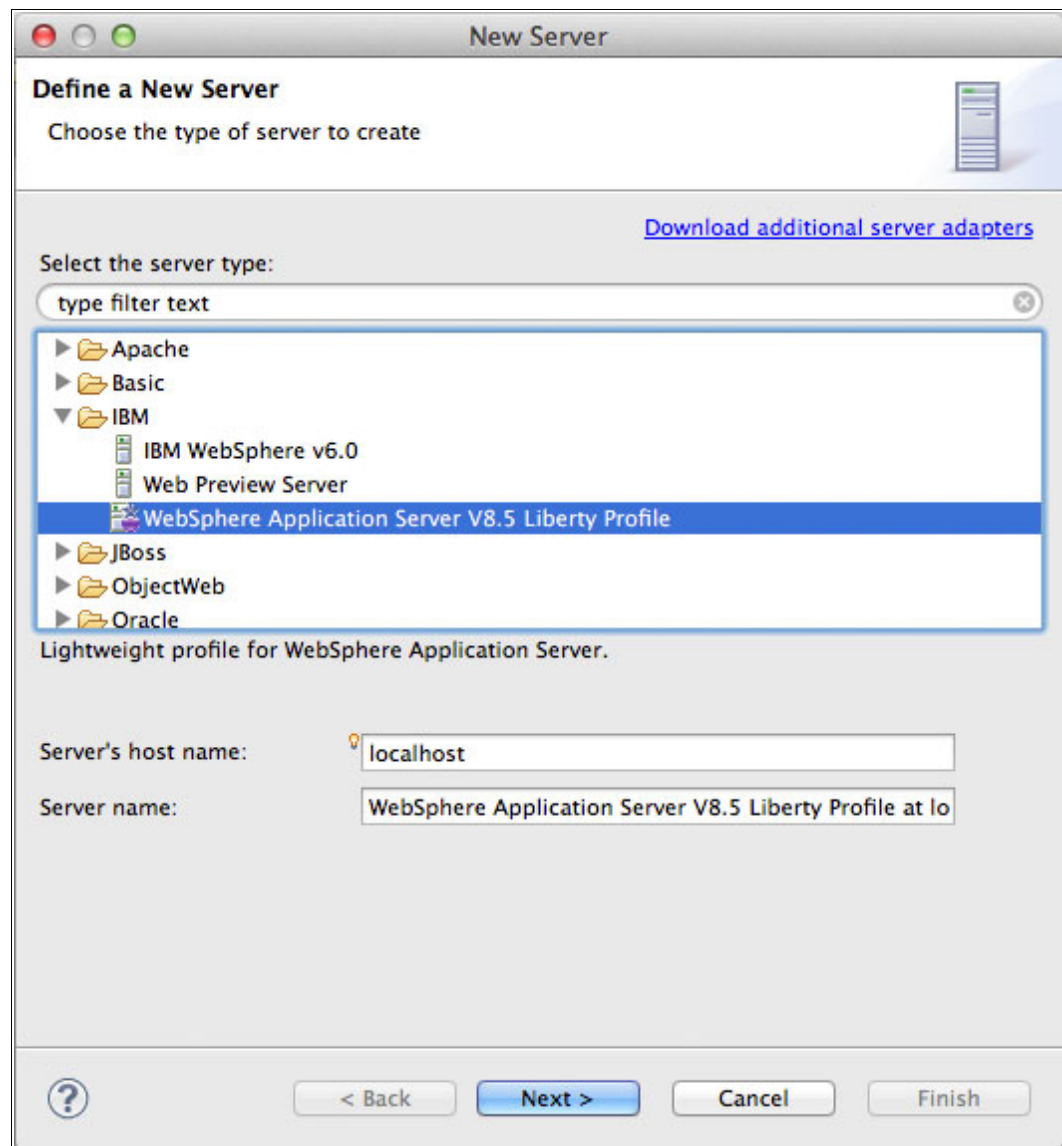


Figure 6-22 Selecting server type

- b. On the Liberty Profile Runtime Environment pane, click **Download or install** link to install the selected runtime environment (Figure 6-23).

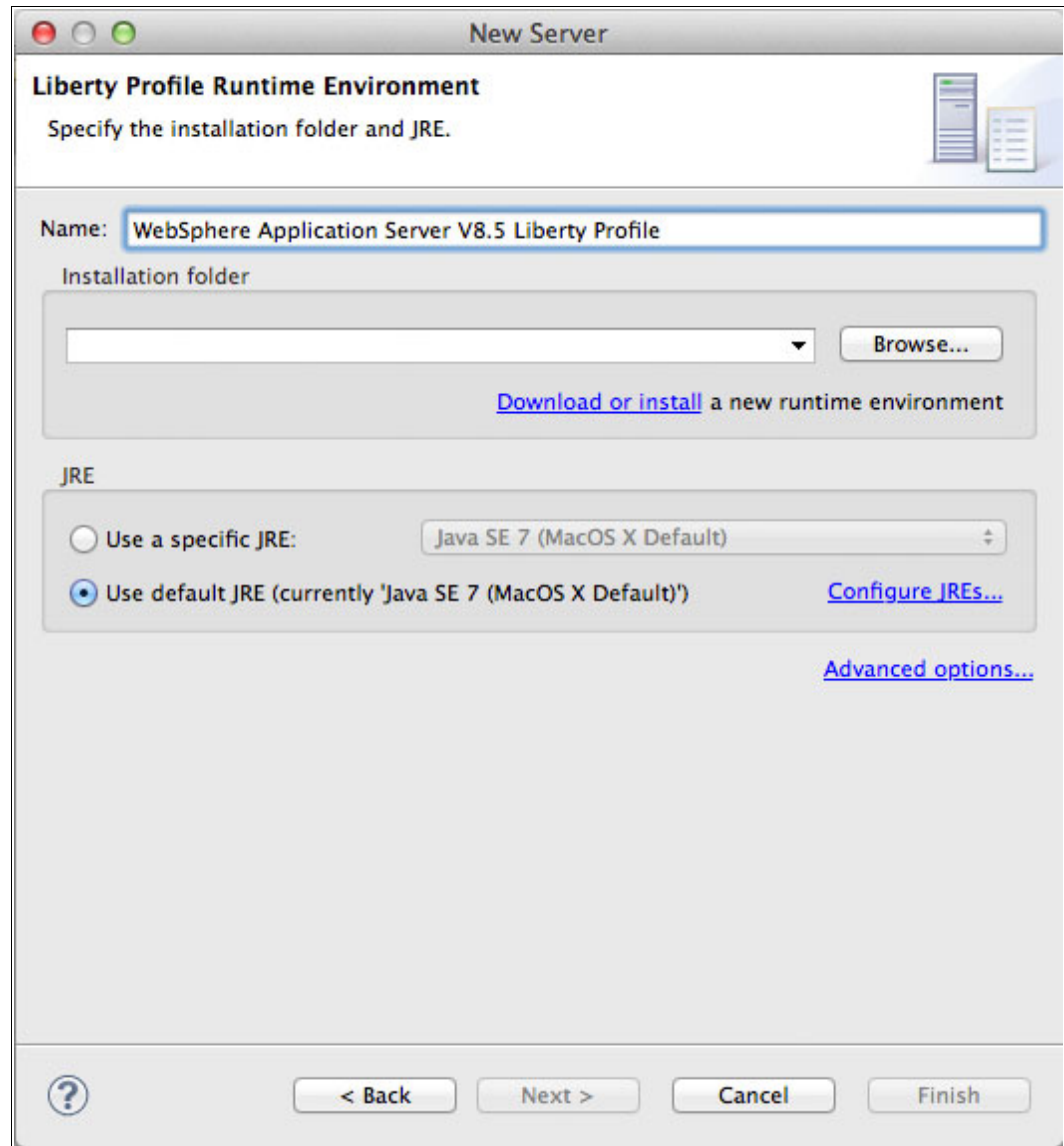


Figure 6-23 Specifying installation directory and JRE

- c. In the new Install Runtime Environment window, select the **Download and install a new runtime environment from** option and then select the IBM WebSphere Application Server V8.5 Liberty Profile download site (Figure 6-24).

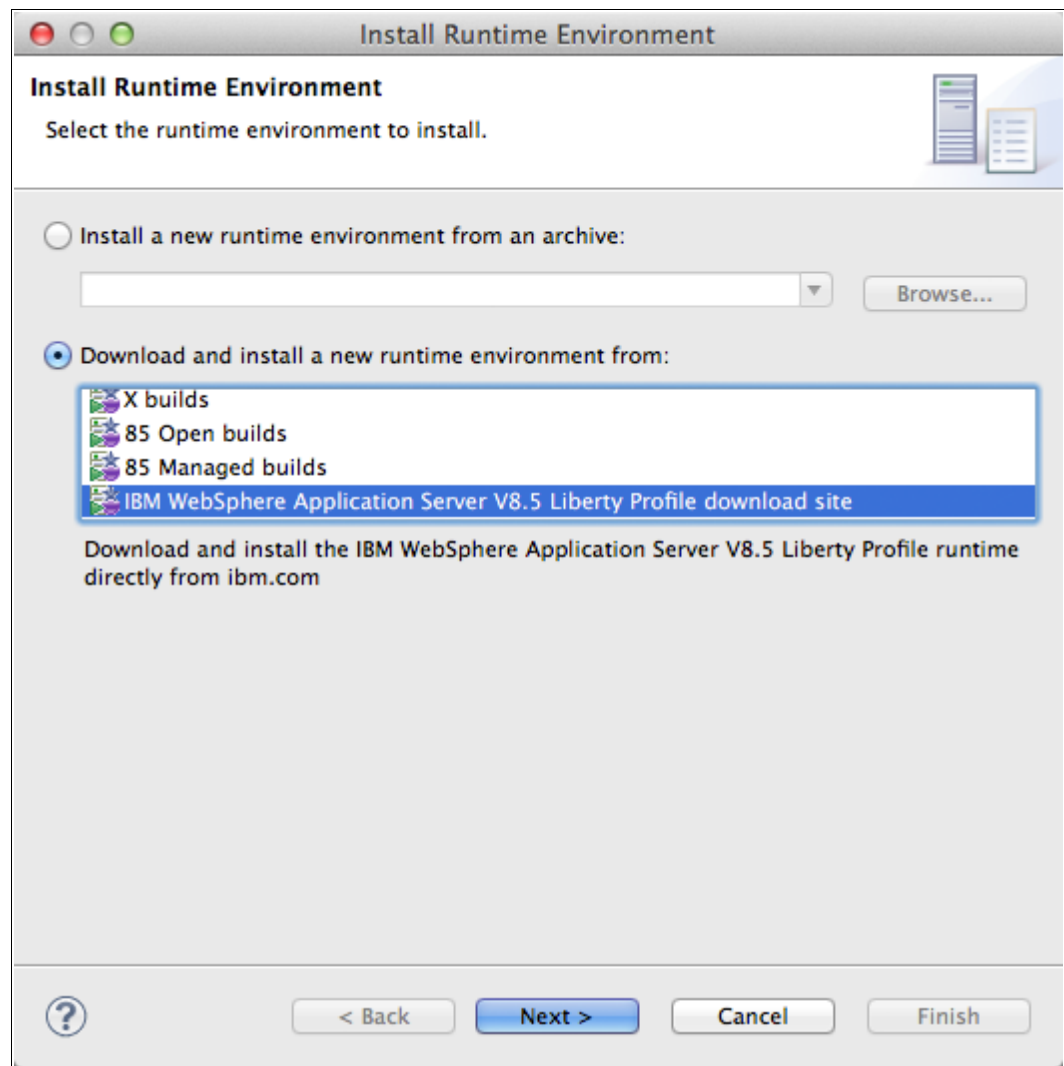


Figure 6-24 Selecting the Liberty Profile download site

- d. Click **Next**.
- e. Review and accept the licensing agreements and then click **Next**.
- f. Designate where to install the runtime environment by specifying an installation folder (for example, /Applications/eclipse/liberty) and then click **Finish** to begin the installation.
- g. When the installation of the new runtime environment is finished, the New Server dialog (Figure 6-23 on page 313) is displayed again, with the installation folder from the previous step displayed in the text field. Now, specify the JRE for the new server. In this example, the authors selected **Use default JRE**.
- h. Click **Next**.

- i. On the New Liberty Profile Server window, specify a server name (in this example, it is defaultServer) and click **Finish**, as shown in Figure 6-25.

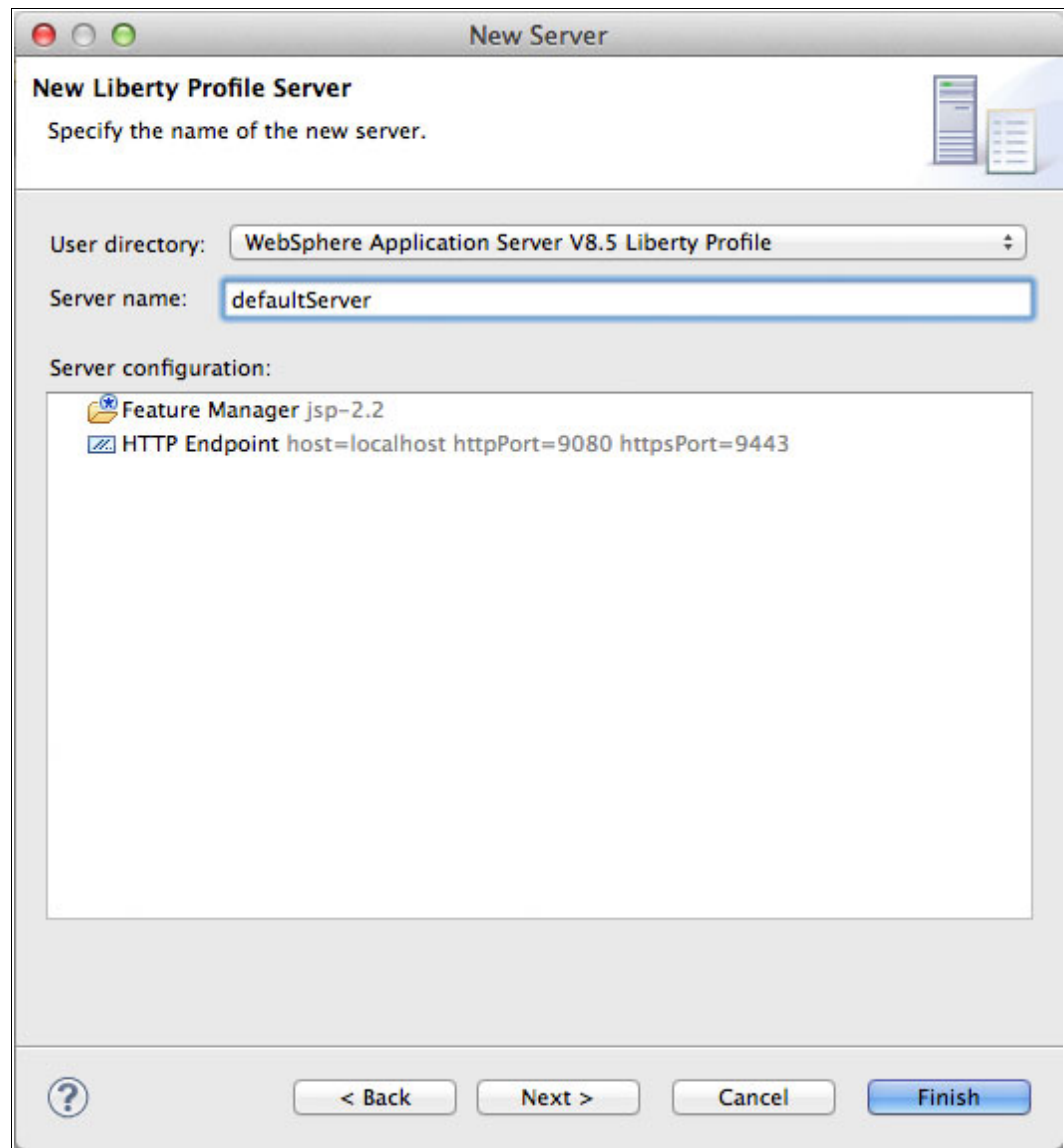


Figure 6-25 Specifying a server name

In the Server view, you will now see a new entry for the Liberty Profile test server.

6.5 Installing IBM WebSphere Cast Iron

IBM WebSphere Cast Iron consists of a server component and a developer studio:

- ▶ The server component is used by IBM Worklight Server to integrate with existing back-end systems or cloud services.
- ▶ The developer studio is used by developers to create new data flows called *orchestrations*. The developer studio has a built-in server to test the orchestrations prior to their being deployed to a IBM WebSphere Cast Iron Hypervisor Edition production server.

6.5.1 Deploying the server component (virtual appliance template)

The IBM WebSphere Cast Iron server component is available as a physical appliance (WebSphere DataPower Cast Iron Appliance XH40) and a virtual appliance (WebSphere Cast Iron Hypervisor Edition). To support the airline scenario presented in this book, the virtual appliance (Hypervisor Edition) is used.

IBM WebSphere Cast Iron Hypervisor Edition (the virtual appliance) is contained within an Open Virtualization Format (OVF) template, so instead of installing the appliance, you deploy the template to a supported hypervisor and the virtual machine is created with the Cast Iron Operating System (CIOS) and VMwareTools already installed. After the OVF template is deployed, you can manage the virtual appliance from the vSphere Client Console tab, as you would manage a physical integration appliance through the command line interface.

Details about how to deploy the OVF template to create the virtual machine is in the IBM WebSphere Cast Iron Version 6.1 Information Center:

http://publib.boulder.ibm.com/infocenter/wci/v6r1m0/topic/com.ibm.websphere.cast_iron.VAuserguide.doc/VA_about_VA.html

Important: IBM Worklight Server must be able to connect to the IBM WebSphere Cast Iron virtual machine in order to use published services. If a security mechanism, such as a firewall or gateway server, protects your internal network entities, you need to ensure that the virtual machine can access the Internet for cloud services integration.

6.5.2 Installing the developer studio

For purposes of this book, the developer studio of IBM WebSphere Cast Iron V6.1 was installed on a machine running Microsoft Windows XP.

Before you start this installation procedure, ensure that you have access to the IBM WebSphere Cast Iron Studio setup file from your installation media. Also confirm that your system meets the requirements detailed at this address:

http://pic.dhe.ibm.com/infocenter/prodguid/v1r0/clarity-reports/report/html/softwareReqsForProductByComponent?deliverableId=1283434088185&duComponent=Desktop_DAF793F0772111E1A7CBA8580925FE36

Install IBM WebSphere Cast Iron Studio on a Microsoft Windows system as follows:

1. Run the Cast Iron Studio setup program from your installation media, shown in Figure 6-26, to start the installation wizard.



Figure 6-26 Cast Iron Studio setup program icon

2. In the Welcome panel (Figure 6-27 on page 317), click **Next**.

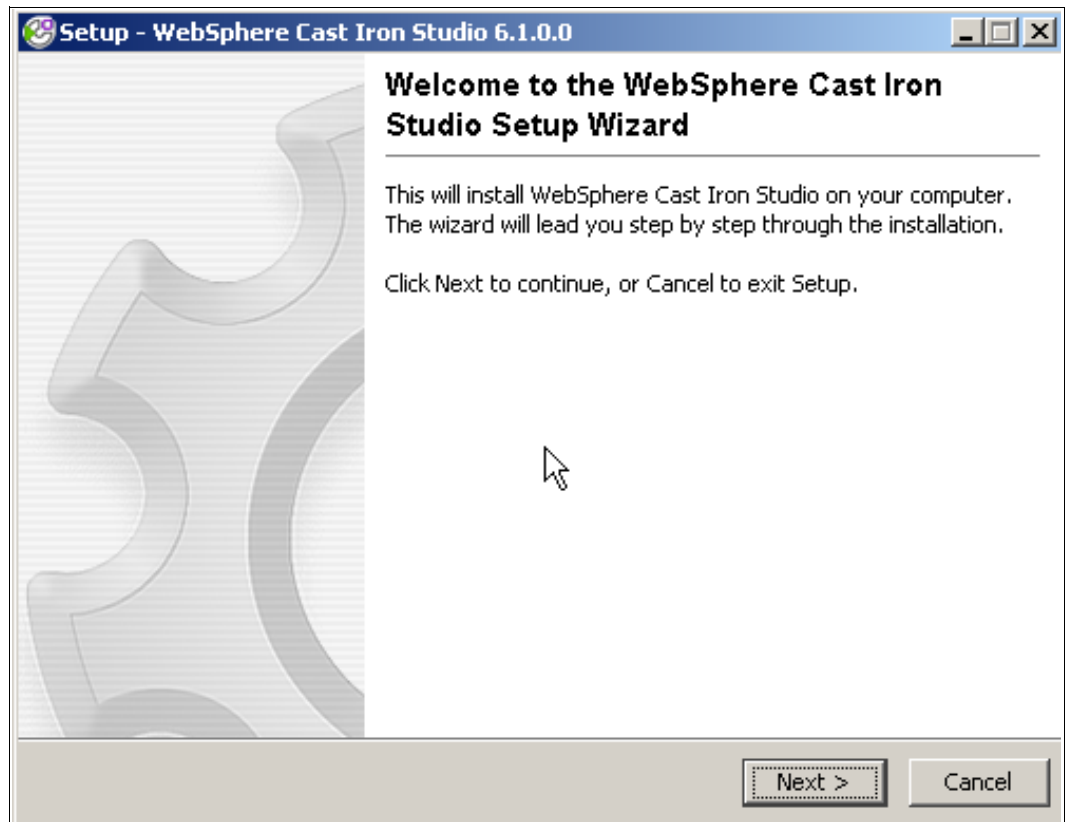


Figure 6-27 WebSphere Cast Iron Studio welcome wizard

3. Read and accept the license agreement and then click **Next**.
4. In the Destination Directory panel, specify the installation path, as shown in Figure 6-28 on page 318, and then click **Next** to begin the installation.

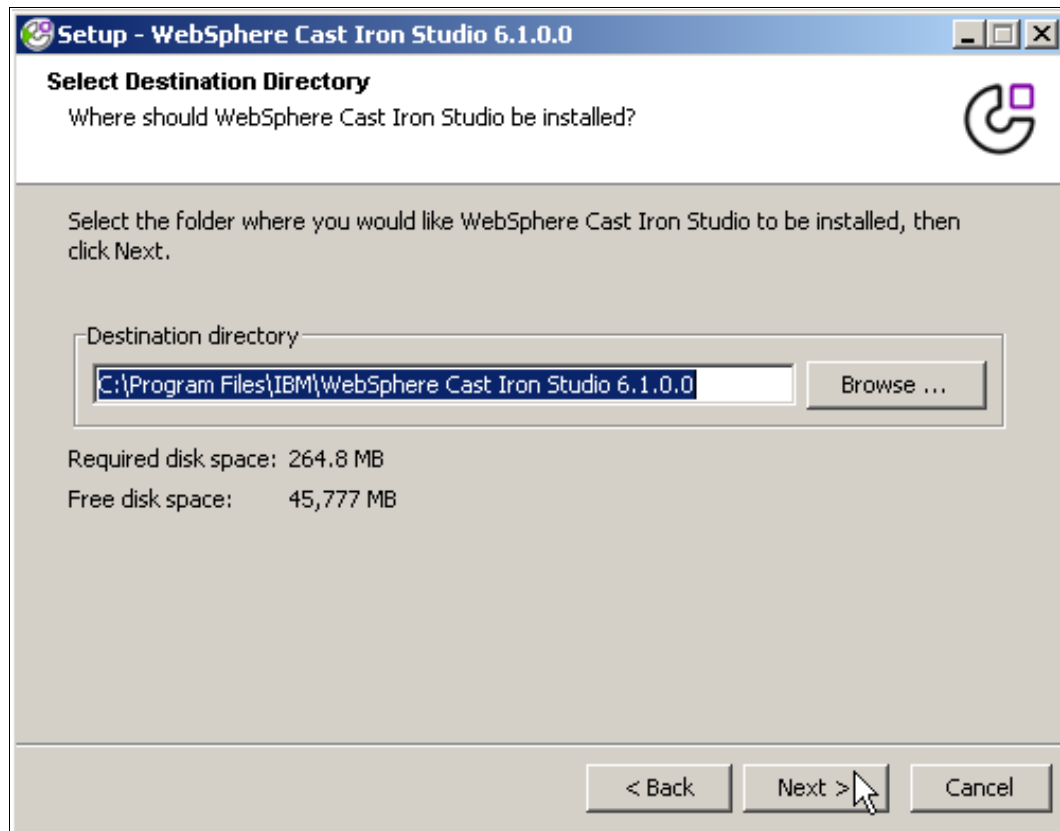


Figure 6-28 Specifying the destination directory

When installation is complete, IBM WebSphere Cast Iron Studio starts automatically (Figure 6-29).

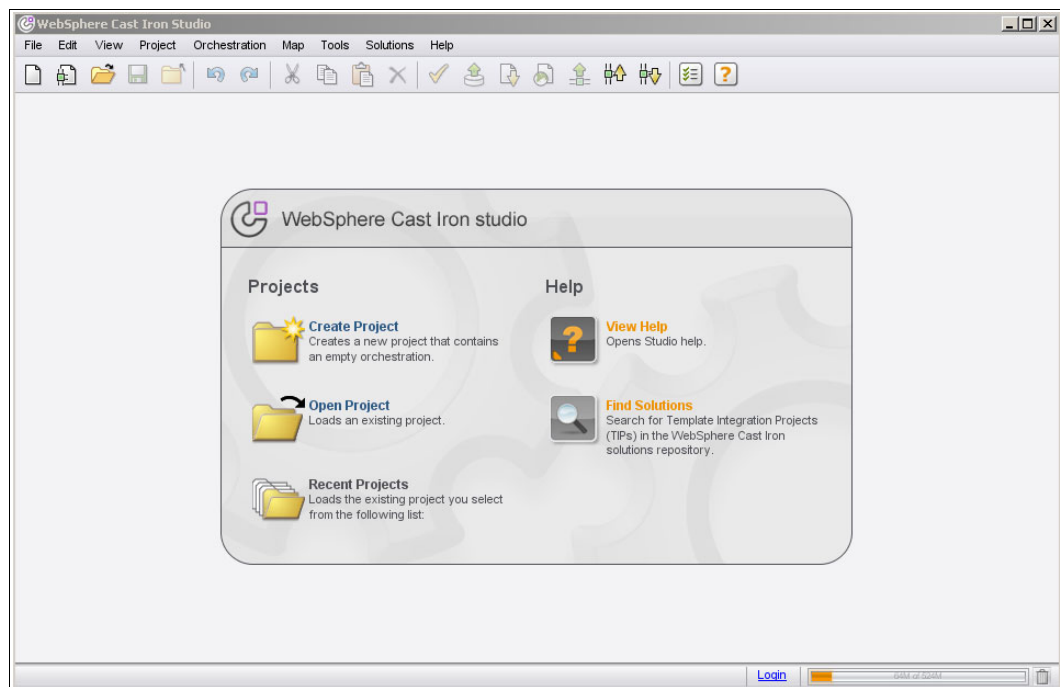


Figure 6-29 IBM WebSphere Cast Iron Studio

6.6 Installing IBM WebSphere eXtreme Scale

IBM WebSphere eXtreme Scale is the caching solution used for Airline Company A's new mobile applications. IBM WebSphere eXtreme Scale Version 8.5 is installed on a system running Red Hat Enterprise Linux Server release 6.2 using IBM Installation Manager.

To install IBM WebSphere eXtreme Scale, use the following steps:

1. Start IBM Installation Manager and add the IBM WebSphere eXtreme Scale installation repository as explained in 6.1, “Installing products with IBM Installation Manager” on page 294.
2. On the main IBM Installation Manager window, click **Install**.
3. Select the appropriate installation package, which in this example is **IBM WebSphere eXtreme Scale in a stand-alone environment** (Figure 6-30), and then click **Next**.

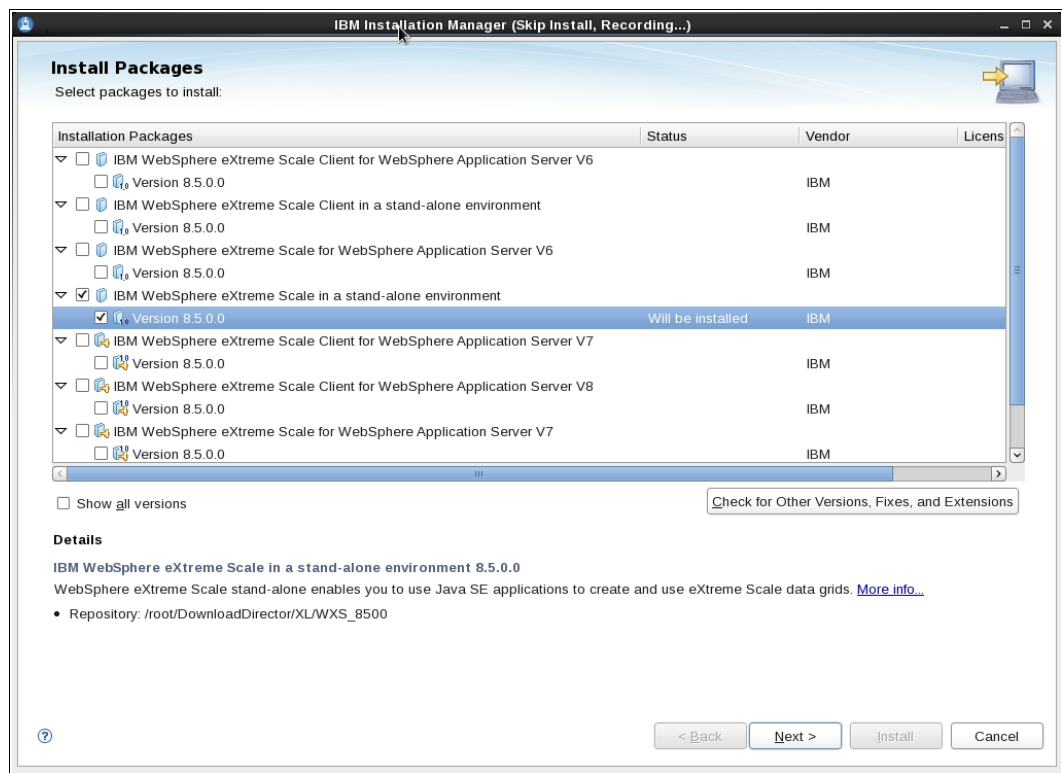


Figure 6-30 Selecting an IBM WebSphere eXtreme Scale server edition to install

4. Proceed through the installation wizard by accepting the default values, providing the appropriate installation directory and architecture details, and accepting the license agreement, until the Feature list (Figure 6-17 on page 308) is displayed.

5. From the Features list, select the **Server**, **Client**, and **Console** features, as shown in Figure 6-31, and then click **Next**.

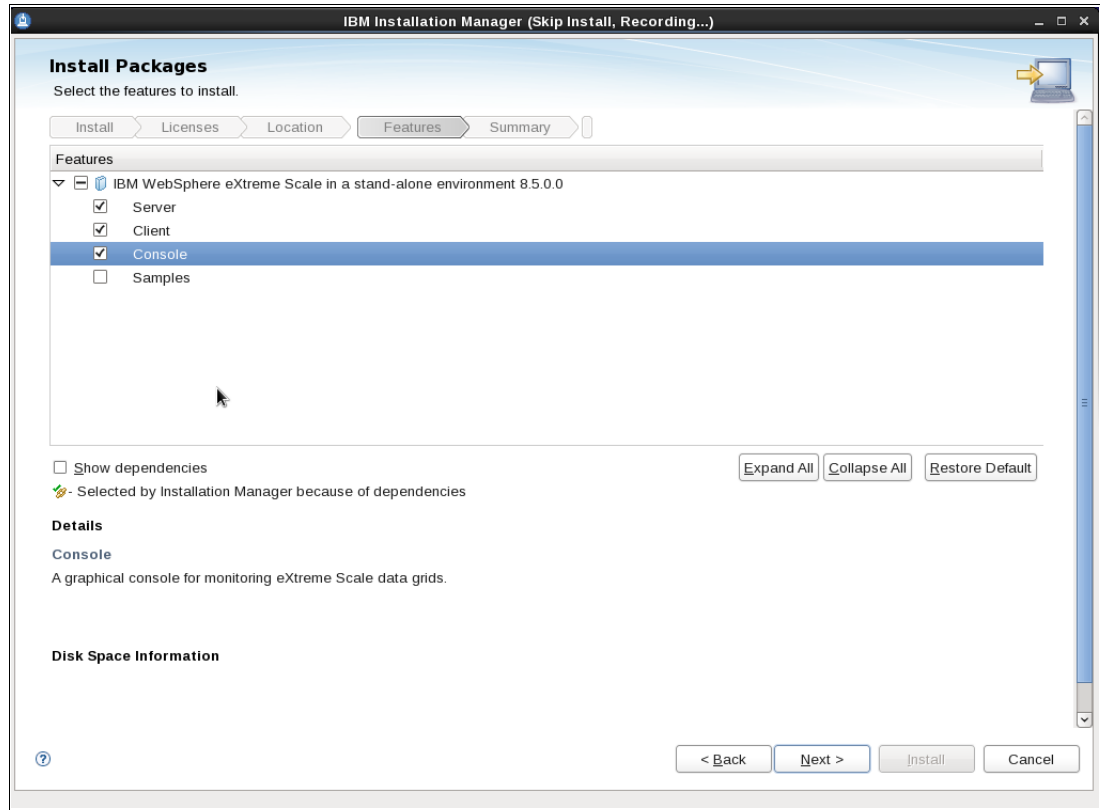


Figure 6-31 Selecting the IBM WebSphere eXtreme Scale features to install

6. Review the summary information and then click **Install** to start the installation procedure.
7. When installation is complete, a confirmation is displayed.

For information about installing IBM WebSphere eXtreme Scale on other platforms or to troubleshoot installation errors, see the IBM WebSphere eXtreme Scale V8.5 Information Center:

http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Fwelcome%2Fwelcome_xs.html



Mobile solution additional source

This appendix provides the source code for various parts of the mobile solution. If you complete the steps in the previous chapters, you have already created these files. They are provided here as a convenience. In addition, an example user registry (used in Chapter 4, “Creating and deploying the mobile solution” on page 59) and the WebSphere eXtreme Scale utility classes (used in Chapter 5, “Expanding the solution to consumers” on page 205) are provided.

The application files are split into three sections:

- ▶ “Worklight Server server.xml file” on page 323
- ▶ “Back-end service source” on page 325
- ▶ “LuggageTracker application source files” on page 335
- ▶ “MyLuggage application source files” on page 350

IBM DOES NOT WARRANT OR REPRESENT THAT THE CODE PROVIDED IS COMPLETE OR UP-TO-DATE. IBM DOES NOT WARRANT, REPRESENT OR IMPLY RELIABILITY, SERVICEABILITY OR FUNCTION OF THE CODE. IBM IS UNDER NO OBLIGATION TO UPDATE CONTENT NOR PROVIDE FURTHER SUPPORT.

ALL CODE IS PROVIDED "AS IS," WITH NO WARRANTIES OR GUARANTEES WHATSOEVER. IBM EXPRESSLY DISCLAIMS TO THE FULLEST EXTENT PERMITTED BY LAW ALL EXPRESS, IMPLIED, STATUTORY AND OTHER WARRANTIES, GUARANTEES, OR REPRESENTATIONS, INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY AND INTELLECTUAL PROPERTY RIGHTS. YOU UNDERSTAND AND AGREE THAT YOU USE THESE MATERIALS, INFORMATION, PRODUCTS, SOFTWARE, PROGRAMS, AND SERVICES, AT YOUR OWN DISCRETION AND RISK AND THAT YOU WILL BE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY RESULT, INCLUDING LOSS OF DATA OR DAMAGE TO YOUR COMPUTER SYSTEM.

IN NO EVENT WILL IBM BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY OR CONSEQUENTIAL DAMAGES OF ANY TYPE WHATSOEVER RELATED TO OR ARISING FROM USE OF THE CODE FOUND HEREIN, WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, LOST SAVINGS, LOSS OF PROGRAMS OR OTHER DATA, EVEN IF IBM

IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS EXCLUSION AND WAIVER OF LIABILITY APPLIES TO ALL CAUSES OF ACTION, WHETHER BASED ON CONTRACT, WARRANTY, TORT OR ANY OTHER LEGAL THEORIES.

THIRD PARTY SOFTWARE IS LICENSED AND DISTRIBUTED TO YOU BY THE **THIRD PARTY** DISTRIBUTORS AND/OR RESPECTIVE COPYRIGHT AND OTHER RIGHT HOLDERS UNDER THEIR TERMS AND CONDITIONS. IBM MAKES NO EXPRESS OR IMPLIED WARRANTIES OR REPRESENTATIONS WITH RESPECT TO SUCH **SOFTWARE** AND PROVIDES NO INDEMNITY FOR SUCH **SOFTWARE**. IBM GRANTS NO EXPRESS OR IMPLIED PATENT OR OTHER LICENSE WITH RESPECT TO **AND** IS NOT LIABLE FOR ANY DAMAGES ARISING OUT OF THE USE OF **SUCH SOFTWARE-**

Worklight Server server.xml file

The server.xml file was modified in 4.5.1, “Managing access to the Application Center” on page 78 to configure the Liberty profile security registry. The basic registry was configured to add users and groups and update the security for the application. The contents of the file is shown in Example A-1 with the added lines highlighted.

Example A-1 Updated server.xml

```
<!--
  Licensed Materials - Property of IBM
  5725-G92 (C) Copyright IBM Corp. 2011, 2012. All Rights Reserved.
  US Government Users Restricted Rights - Use, duplication or
  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
-->
<server description="IBM Worklight Server and Application Center">
  <!-- Enable features -->
  <featureManager>
    <feature>jsp-2.2</feature>

    <!-- Begin of features added by IBM Worklight installer. -->
    <feature>ssl-1.0</feature>
    <feature>servlet-3.0</feature>
    <feature>jdbc-4.0</feature>
    <feature>security-1.0</feature>
    <feature>appSecurity-1.0</feature>
    <!-- End of features added by IBM Worklight installer. -->
  </featureManager>

  <httpEndpoint id="defaultHttpEndpoint"
    host="*"
    httpPort="9080"
    httpsPort="9443" >
    <!-- Begin of options added by IBM Worklight installer. -->
    <tcpOptions soReuseAddr="true"/>
    <!-- End of options added by IBM Worklight installer. -->
  </httpEndpoint>

  <!-- Begin of configuration added by IBM Worklight installer. -->

  <!-- Declare the IBM Worklight Server application. -->
  <application id="worklight" name="worklight" location="worklight.war"
type="war">
    <classloader delegation="parentLast">
      <commonLibrary>
        <fileset dir="{shared.resource.dir}/lib"
includes="worklight-jee-library.jar"/>
      </commonLibrary>
    </classloader>
  </application>

  <!-- Declare the IBM Application Center application. -->
  <application id="applicationcenter" name="applicationcenter"
location="applicationcenter.war" type="war">
    <application-bnd>
```

```

        <security-role name="appcenteradmin">
            <group name="appcentergroup"/>
        </security-role>
        <security-role name="appcenteruser">
            <group name="appcenterusergroup"/>
        </security-role>
    </application-bnd>
</application>

<!-- Declare web container custom properties for the IBM Worklight Server
application. -->
<webContainer invokeFlushAfterService="false"/>

<!-- Declare the user registry for the IBM Application Center. -->
<!-- Declare the user registry for the IBM Application Center. -->
<basicRegistry id="applicationcenter-registry" realm="ApplicationCenter">
    <!-- The user "appcenteradmin" has special privileges within the IBM
Application Center. -->
    <user name="appcenteradmin" password="admin"/>
    <!-- The other users have normal privileges. -->
    <user name="demo" password="demo"/>
    <user name="CSR1" password="csr1"/>
    <user name="CSR2" password="csr2"/>
    <user name="CSR3" password="csr3"/>
    <user name="CSR4" password="csr4"/>
    <user name="CSR5" password="csr5"/>
    <user name="CSR6" password="csr6"/>
    <user name="Tester1" password="tester1"/>
    <user name="Tester2" password="tester2"/>
    <group name="appcentergroup">
        <member name="appcenteradmin"/>
    </group>
    <group name="appcenterusergroup">
        <member name="demo"/>
        <member name="CSR1"/>
        <member name="CSR2"/>
        <member name="CSR3"/>
        <member name="CSR4"/>
        <member name="CSR5"/>
        <member name="CSR6"/>
        <member name="Tester1"/>
        <member name="Tester2"/>
    </group>
</basicRegistry>

<!-- Declare the jar file for Derby with the "embedded" deployment option. -->
<library id="DerbyLib">
    <fileset dir="${shared.resource.dir}/derby" includes="derby.jar"/>
</library>

<!-- Declare the IBM Worklight Server database. Used through property
wl.db.jndi.name.
If you change this declaration to refer to a different kind of data base,
you have to update the property wl.db.type in the file
worklight.properties inside the file worklight.war. -->

```

```

        <dataSource id="WorklightDS" jndiName="jdbc/WorklightDS">
            <jdbcDriver libraryRef="DerbyLib"/>
            <properties.derby.embedded databaseName="C:\Documents and Settings\All
Users\Application Data\IBM\Worklight\derby\WRKLGHT" user="WORKLIGHT"/>
        </dataSource>

<!-- Declare the IBM Worklight Server Reports database. Used through property
wl.reports.db.jndi.name. If you change this declaration to refer to a
different kind of data base, you have to update the property
wl.reports.db.type in the file worklight.properties inside the file
worklight.war. -->
        <dataSource id="WorklightReportsDS" jndiName="jdbc/WorklightReportsDS">
            <jdbcDriver libraryRef="DerbyLib"/>
            <properties.derby.embedded databaseName="C:\Documents and Settings\All
Users\Application Data\IBM\Worklight\derby\WLREPORT" user="WORKLIGHT"/>
        </dataSource>

        <!-- Declare the IBM Application Center database. -->
        <dataSource id="AppCenterDS" jndiName="jdbc/AppCenterDS">
            <jdbcDriver libraryRef="DerbyLib"/>
            <properties.derby.embedded databaseName="C:\Documents and Settings\All
Users\Application Data\IBM\Worklight\derby\APPCNTR" user="APPCENTER"/>
        </dataSource>

        <!-- End of configuration added by IBM Worklight installer. -->

</server>

```

Back-end service source

The LuggageTracker and MyLuggage applications use adapters that call into back-end services. For the purpose of this book, these back-end services are simulated. The source files listed in this section were used to simulate the back-end services and can be used to replicate the environment that was used during the writing of this book.

Java package com.ibm.itso.saw210.beans

This package contains the classes representing the basic business objects, using annotations to generate the corresponding XML messages used in the back-end services.

Address.java

```

package com.ibm.itso.saw210.beans;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="address")
public class Address {

    String name;
    String addressLine1;
    String addressLine2;
    String city;
    String state;
}

```

```

String zipCode;
String phoneNumber;
String comments;

public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getAddressLine1() {
    return addressLine1;
}
public void setAddressLine1(String addressLine1) {
    this.addressLine1 = addressLine1;
}
public String getAddressLine2() {
    return addressLine2;
}
public void setAddressLine2(String addressLine2) {
    this.addressLine2 = addressLine2;
}
public String getCity() {
    return city;
}
public void setCity(String city) {
    this.city = city;
}
public String getState() {
    return state;
}
public void setState(String state) {
    this.state = state;
}
public String getZipCode() {
    return zipCode;
}
public void setZipCode(String zipCode) {
    this.zipCode = zipCode;
}
public String getPhoneNumber() {
    return phoneNumber;
}
public void setPhoneNumber(String phoneNumber) {
    this.phoneNumber = phoneNumber;
}
public String getComments() {
    return comments;
}
public void setComments(String comments) {
    this.comments = comments;
}
}

```


Luggage.java

```
package com.ibm.itso.saw210.beans;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="luggage")

public class Luggage {

    public static final String STATUS_ON_TIME = "On Time";
    public static final String STATUS_DELAYED = "Delayed";
    public static final String STATUS_LOST = "Lost";
    public static final String STATUS_PICKED = "Picked";

    private String trackingId;
    private String ownerName;
    private String flightNumber;
    private String currentLocation;
    private String nextLocation;
    private String finalDestination;
    private String status;
    private Address deliveryAddress;
    private String comments;
    private String lastUpdate;

    public String getTrackingId() {
        return trackingId;
    }

    public void setTrackingId(String trackingId) {
        this.trackingId = trackingId;
    }

    public String getOwnerName() {
        return ownerName;
    }

    public void setOwnerName(String ownerName) {
        this.ownerName = ownerName;
    }

    public String getFlightNumber() {
        return flightNumber;
    }

    public void setFlightNumber(String flightNumber) {
        this.flightNumber = flightNumber;
    }

    public String getCurrentLocation() {
        return currentLocation;
    }

    public void setCurrentLocation(String currentLocation) {
        this.currentLocation = currentLocation;
    }

    public String getNextLocation() {
        return nextLocation;
    }
}
```

```

    public void setNextLocation(String nextLocation) {
        this.nextLocation = nextLocation;
    }

    public String getFinalDestination() {
        return finalDestination;
    }

    public void setFinalDestination(String finalDestination) {
        this.finalDestination = finalDestination;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public Address getDeliveryAddress() {
        return deliveryAddress;
    }

    public void setDeliveryAddress(Address deliveryAddress) {
        this.deliveryAddress = deliveryAddress;
    }

    public String getComments() {
        return comments;
    }

    public void setComments(String comments) {
        this.comments = comments;
    }

    public String getLastUpdate() {
        return lastUpdate;
    }

    public void setLastUpdate(String lastUpdate) {
        this.lastUpdate = lastUpdate;
    }

    public String toString() {
        return
        "id="+trackingId+";ownerName="+ownerName+";flightNumber="+flightNumber+";status="+status
        ;//+";deliveryAddress="+deliveryAddress+";comments="+comments;
    }
}

```

com.ibm.itso.saw210.dao

This package represents the data access layer. In our sample application, the Database class simulates the data managed by an in-memory database. The DatabaseLoader class loads some sample data to be used for testing purposes.

Database.java

```
package com.ibm.itso.saw210.dao;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.TreeSet;

import com.ibm.itso.saw210.beans.Luggage;
import com.ibm.itso.saw210.util.Formatter;

public class Database {

    private static Database instance = null;
    private HashMap<String, Luggage> luggage;

    private Database() {
        luggage = new HashMap<String, Luggage>();
    }

    public static void load() {
        DatabaseLoader.loadLuggage();
    }

    private static Database getInstance() {
        if (instance==null) {
            instance = new Database();
        }

        return instance;
    }

    public static void insertLuggage(Luggage luggage){
        luggage.setLastUpdate(Formatter.format(new Date()));
        getInstance().luggage.put(luggage.getTrackingId(), luggage);
    }

    public static void deleteLuggage(String luggageId){
        getInstance().luggage.remove(luggageId);
    }

    public static Luggage selectLuggage(String luggageId) {
        return getInstance().luggage.get(luggageId);
    }

    public static ArrayList<Luggage> selectAllLuggage(){
        ArrayList<Luggage> out = new ArrayList<Luggage>();
        TreeSet<String> keys = new TreeSet<String>(instance.luggage.keySet());
        for (String key : keys) {
            out.add(getInstance().luggage.get(key));
        }
        return out;
    }
}
```

DatabaseLoader.java

```
package com.ibm.itso.saw210.dao;

import com.ibm.itso.saw210.beans.Address;
import com.ibm.itso.saw210.beans.Luggage;

public class DatabaseLoader {

    public static void loadLuggage() {

        Luggage luggage3 = new Luggage();
        luggage3.setTrackingId("007986");
        luggage3.setOwnerName("Hossam Katory");
        luggage3.setFlightNumber("XX 666");
        luggage3.setStatus(Luggage.STATUS_DELAYED);
        luggage3.setComments("Luggage will be forwarded with next flight");
        luggage3.setCurrentLocation("PVG");
        luggage3.setNextLocation("DXB");
        luggage3.setFinalDestination("CAI");
        Address address3 = new Address();
        address3.setName("Hossam Katory");
        address3.setAddressLine1("123 Haram St");
        address3.setAddressLine2("Alharam 3B");
        address3.setCity("Giza");
        address3.setState("CAI");
        address3.setZipCode("99999");
        address3.setPhoneNumber("202-3655-1234");
        address3.setComments("Deliver between 9AM to 6PM");
        luggage3.setDeliveryAddress(address3);

        Database.insertLuggage(luggage3);

    }
}
```

com.ibm.itso.saw210.facade

This package represents the access to the business logic layer of the application. In our sample it only contains one class with methods to be called when invoking the services.

LuggageFacade.java

```
package com.ibm.itso.saw210.facade;

import com.ibm.itso.saw210.beans.Address;
import com.ibm.itso.saw210.beans.Luggage;
import com.ibm.itso.saw210.dao.Database;

public class LuggageFacade {

    public static Luggage getLuggageInformation(String luggageId) {
        return Database.selectLuggage(luggageId);
    }
}
```

```

        public static Luggage addDeliveryInformation(String luggageId, Address
deliveryAddress) {

            Luggage luggage = Database.selectLuggage(luggageId);
            if (luggage != null) {
                luggage.setDeliveryAddress(deliveryAddress);

                Database.insertLuggage(luggage);
            }
            return luggage;
        }
    }
}

```

com.ibm.itso.saw210.services.rest

This package contains the services implementation. The `AirlineApplication` class prepares the JAX-RS engine to handle services requests by registering the `LuggageServiceImpl` class as a service implementation. `LuggageService` is the interface used to declare the available service methods to retrieve and update luggage information. `LuggageServiceImpl` is the concrete implementation of `LuggageService`, calling the corresponding methods of the `LuggageFacade` to invoke business logic.

AirlineApplication.java

```

package com.ibm.itso.saw210.services.rest;

import java.util.HashSet;
import java.util.Set;
import javax.ws.rs.core.Application;

public class AirlineApplication extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        Set<Class<?>> classes = new HashSet<Class<?>>();
        classes.add(LuggageServiceImpl.class);
        return classes;
    }
}

```

LuggageService.java

```

package com.ibm.itso.saw210.services.rest;

import javax.ws.rs.Consumes;
import javax.ws.rs.FormParam;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.xml.bind.JAXBElement;
import com.ibm.itso.saw210.beans.Luggage;

@Path("/luggage")

```

```

public interface LuggageService {

    @GET
    @Path(value="/{luggageId}")
    @Produces(MediaType.APPLICATION_XML)
    public Luggage[] getLuggageInformation(@PathParam(value="luggageId") String
luggageId);

    @POST
    @Consumes(MediaType.APPLICATION_XML)
    @Produces(MediaType.APPLICATION_XML)
    public Luggage updateDeliveryInformation(JAXBElement<Luggage> luggage);

    @POST
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    @Produces(MediaType.APPLICATION_XML)
    public Luggage updateDeliveryInformation(@FormParam(value="trackingId")
String trackingId,
        @FormParam(value="name")String name,
        @FormParam(value="addressLine1")String addressLine1,
        @FormParam(value="addressLine2")String addressLine2,
        @FormParam(value="city")String city,
        @FormParam(value="state")String state,
        @FormParam(value="zipCode")String zipCode,
        @FormParam(value="phoneNumber")String phoneNumber,
        @FormParam(value="comments")String comments) ;
}

```

LuggageServiceImpl.java

```

package com.ibm.itso.saw210.services.rest;

import javax.ws.rs.Path;
import javax.xml.bind.JAXBElement;
import com.ibm.itso.saw210.beans.Address;
import com.ibm.itso.saw210.beans.Luggage;
import com.ibm.itso.saw210.facade.LuggageFacade;

@Path("/luggage")
public class LuggageServiceImpl implements LuggageService {

    public Luggage[] getLuggageInformation(String luggageId) {

        Luggage luggage = LuggageFacade.getLuggageInformation(luggageId);
        if (luggage != null) {
            Luggage[] out = new Luggage[1];
            out[0] = luggage;
            return out;
        }
        else {
            return null;
        }
    }

    public Luggage updateDeliveryInformation(String luggageId, String name,
        String addressLine1, String addressLine2, String city, String state,

```

```

        String zipCode, String phoneNumber, String comments) {
    Address deliveryAddress = new Address();
    deliveryAddress.setName(name);
    deliveryAddress.setAddressLine1(addressLine1);
    deliveryAddress.setAddressLine2(addressLine2);
    deliveryAddress.setCity(city);
    deliveryAddress.setState(state);
    deliveryAddress.setZipCode(zipCode);
    deliveryAddress.setPhoneNumber(phoneNumber);
    deliveryAddress.setComments(comments);

    return updateDatabase(luggageId, deliveryAddress);
}

public Luggage updateDeliveryInformation(JAXBElement<Luggage> luggage) {
    Luggage temp = luggage.getValue();
    return updateDatabase(temp.getTrackingId(), temp.getDeliveryAddress());
}

private Luggage updateDatabase(String luggageId, Address deliveryAddress) {
    return LuggageFacade.addDeliveryInformation(luggageId, deliveryAddress);
}
}

```

com.ibm.itso.saw210.servlet

This package contains one servlet to load the database sample data in memory on application startup.

Loader.java

```

package com.ibm.itso.saw210.servlet;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;

import com.ibm.itso.saw210.dao.Database;

public class Loader extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Override
    public void init() throws ServletException {
        super.init();

        Database.load();
    }
}

```

com.ibm.itso.saw210.util

This package contains a formatting utility class.

Formatter.java

```
package com.ibm.itso.saw210.util;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Formatter {

    private static SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
    public static String format(Date date) {
        return sdf.format(date);
    }

    public static Date parse(String date) {
        try {
            return sdf.parse(date);
        } catch (ParseException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Deployment descriptor

The web.xml file is the deployment descriptor of the web application. Some additional configuration was done by indicating the JAX-RS engine implementation. Any JAX-RS engine may be used; in our case, we used the one included in WebSphere Liberty Profile available as a Project Facet on Eclipse. The application class declaring the available services must be indicated as an init-param.

```
<servlet>
  <description>
    JAX-RS Tools Generated - Do not modify</description>
  <servlet-name>JAX-RS Servlet</servlet-name>
  <servlet-class>com.ibm.websphere.jaxrs.server.IBMRestServlet</servlet-class>
  <init-param>
    <param-name>javax.ws.rs.Application
    </param-name>
    <param-value>com.ibm.itso.saw210.services.rest.AirlineApplication
    </param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```


LuggageTracker application source files

The LuggageTracker mobile application is the first mobile application written by Airline Company A, as described in Chapter 4, “Creating and deploying the mobile solution” on page 59. The source files provided here are the complete files, which you already created by following the instructions within that chapter. They are provided here as a reference.

Business services adapter files

The following files make up the business services adapter, created in 4.7.3, “Creating the business services adapter” on page 99. The BusinessServicesAdapter is part of the AirlineAdapters project in Worklight Studio.

Source for BusinessServicesAdapter.xml

The BusinessServicesAdapter.xml file contains the adapter definition using the <wl:adapter> tag. It defines the type of connection (in this example, HTTP) and also the server name and port to connect to the back end. This file is shown in Example A-2. The **server_name** and **server_port** should match your individual environment.

Example A-2 BusinessServicesAdapter.xml contents

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--
    Licensed Materials - Property of IBM
    5725-G92 (C) Copyright IBM Corp. 2011, 2012. All Rights Reserved.
    US Government Users Restricted Rights - Use, duplication or
    disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
-->
<wl:adapter name="BusinessServicesAdapter"
  xmlns:wl="http://www.worklight.com/integration"
  xmlns:http="http://www.worklight.com/integration/http"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <displayName>BusinessServicesAdapter</displayName>
  <description>Provides access to enterprise business services</description>
  <connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>http</protocol>
      <domain>server_name</domain>
      <port>server_port</port>
    </connectionPolicy>
    <loadConstraints maxConcurrentConnectionsPerNode="2"/>
  </connectivity>

  <procedure name="getLuggageInformation" audit="true"/>
  <procedure name="addLuggageDeliveryInformation" audit="true"/>
</wl:adapter>
```

Source for BusinessServicesAdapter-impl.js

The contents of the business services adapter JavaScript implementation file (BusinessServicesAdapter-impl.js) is shown in Example A-3. This file contains functions that are called when the adapter procedures (defined in BusinessServicesAdapter.xml) are invoked. These functions interact with the back-end services through invokeHttp() method calls.

Example A-3 JavaScript source contents for BusinessServicesAdapter-impl.js

```
/*
 * Licensed Materials - Property of IBM
 * 5725-G92 (C) Copyright IBM Corp. 2011, 2012. All Rights Reserved.
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */

function getLuggageInformation(luggageId) {
    var input = {
        method : 'get',
        returnerContentType : 'xml',
        path : 'AirlineREST/jaxrs/luggage/'+luggageId,
        transformation : {
            type : 'xslFile',
            xslFile : 'filtered.xsl'
        }
    };

    return WL.Server.invokeHttp(input);
}

function addLuggageDeliveryInformation(lugId, ownerName, addLine1, addLine2,
cityName, stateName, zip, phone, deliveryComments) {
    var input = {
        method : 'post',
        returnerContentType : 'xml',
        senderContentType : 'xml',
        path : 'AirlineREST/jaxrs/luggage/',
        parameters : {lugId : lugId,
            name : ownerName,
            addressLine1 : addLine1,
            addressLine2 : addLine2,
            city : cityName,
            state : stateName,
            zipCode : zip,
            phoneNumber : phone,
            comments : deliveryComments
        }
    };

    return WL.Server.invokeHttp(input);
}
```

Source for filtered.xsl

The `filtered.xsl` stylesheet file (shown in Example A-4) is used to filter unnecessary data that is returned from the back-end enterprise business services. Because our back-end services were written specifically for the purposes of this book, the transform itself is not necessary. In a real-world scenario, most existing business services will return more data than needed for the application so this example was included to show how the data transformation might be done.

Example A-4 Source contents of filtered.xsl file

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:h="http://www.w3.org/1999/xhtml">
  <xsl:output method="text"/>
  <xsl:template match="/luggage">
    {
      'luggage': {
        'trackingId' : '<xsl:value-of select="trackingId"/>',
        'ownerName' : '<xsl:value-of select="ownerName"/>',
        'status' : '<xsl:value-of select="status"/>',
        'flightNumber' : '<xsl:value-of select="flightNumber"/>',
        'currentLocation' : '<xsl:value-of select="currentLocation"/>',
        'nextLocation' : '<xsl:value-of select="nextLocation"/>',
        'finalDestination' : '<xsl:value-of select="finalDestination"/>',
        'comments' : '<xsl:value-of select="comments"/>',
        'lastUpdate' : '<xsl:value-of select="lastUpdate"/>',
        'deliveryAddress' : {
          'addressLine1' : '<xsl:value-of
select="deliveryAddress/addressLine1"/>',
          'addressLine2' : '<xsl:value-of
select="deliveryAddress/addressLine2"/>',
          'city' : '<xsl:value-of select="deliveryAddress/city"/>',
          'comments' : '<xsl:value-of select="deliveryAddress/comments"/>',
          'name' : '<xsl:value-of select="deliveryAddress/name"/>',
          'phoneNumber' : '<xsl:value-of
select="deliveryAddress/phoneNumber"/>',
          'state' : '<xsl:value-of select="deliveryAddress/state"/>',
          'zipCode' : '<xsl:value-of select="deliveryAddress/zipCode"/>'
        }
      }
    }
  </xsl:template>
</xsl:stylesheet>
```

Authentication adapter files

The following files are part of the LuggageTracker adapter, which handles authentication, created in 4.7.2, “Creating the authentication adapter” on page 91. LuggageTrackerAdapter is part of the AirlineAdapters project in Worklight Studio.

Source for LuggageTrackerAdapter.xml

The `LuggageTrackerAdapter.xml` file contains the adapter definition using the `<wl:adpater>` tag. It defines the type of connection (in this example, HTTP) and also the server name and

port to connect to on the back end. This file is shown in Example A-5. The **server_name** and **server_port** should match your individual environment.

Example A-5 LuggageTrackerAdapter.xml source

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Licensed Materials - Property of IBM
    5725-G92 (C) Copyright IBM Corp. 2011, 2012. All Rights Reserved.
    US Government Users Restricted Rights - Use, duplication or
    disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
-->
<wl:adapter name="LuggageTrackerAdapter"
  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>LuggageTrackerAdapter</displayName>
  <description>LuggageTrackerAdapter</description>
  <connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>http</protocol>
      <domain>server_name </domain>
      <port>server_port </port>
    </connectionPolicy>
    <loadConstraints maxConcurrentConnectionsPerNode="2" />
  </connectivity>

  <procedure name="submitAuthentication" audit="true"/>
  <procedure name="onLogout" audit="true"/>
</wl:adapter>
```

Source for LuggageTrackerAdapter-impl.js

The LuggageTrackerAdapter is the adapter that is used for server-side authentication. It is called by the client-side authentication to validate the credentials provided by the application user. Example A-6 shows the contents of the implementation JavaScript file (LuggageTrackerAdapter-impl.js).

Example A-6 LuggageTrackerAdapter-impl.js source

```
/*
 * Licensed Materials - Property of IBM
 * 5725-G92 (C) Copyright IBM Corp. 2006, 2012. All Rights Reserved.
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */

function onAuthRequired(headers, errorMessage){
  WL.Logger.debug("onAuthRequired");
  errorMessage = errorMessage ? errorMessage : null;

  return {
    isLoginSuccessful: false,
    authRequired: true,
    errorMessage: errorMessage
  };
};
```

```

}

function submitAuthentication(username, password){
    WL.Logger.debug("AUth server call submitted");
    WL.Logger.debug("user:" + username);
    WL.Logger.debug("pass:" + password);
    if (username=="worklight" && password === "worklight"){
        WL.Logger.debug("AUth call submitted");
        var userIdentity = {
            userId: username,
            displayName: username,
            attributes: {
                foo: "bar"
            }
        };

        WL.Server.setActiveUser("LuggageTrackerRealm", userIdentity);

        return {
            isLoginSuccessful:true,
            authRequired: false
        };
    }
    WL.Server.setActiveUser("LuggageTrackerRealm", null);
    return onAuthRequired(null, "Invalid login credentials");
}

function onLogout(){
    WL.Server.setActiveUser("LuggageTrackerRealm", null);
    WL.Logger.debug("Logged out");
}

```

User registry

As 4.7.1, “Setting up the adapter project” on page 88 describes, a user registry is used to authenticate the user credentials during the login process of the mobile application. In a real-life scenario, this will go to an existing user registry for authentication. For the purposes of this book, a file-based user registry was created to simulate an actual user registry.

Source for UserRegistry.java

The UserRegistry class is an interface class. It can be used to implement multiple types of user registries. As shown in Example A-7, the interface contains a single authenticate() method.

Example A-7 UserRegistry interface source

```

package com.ibm.itso.saw210.security;

public interface UserRegistry {
    boolean authenticate (String userid, String password);
}

```

Source for UserRegistryFactory.java

The UserRegistryFactory class is a wrapper class that handles integration with the back-end security mechanism. With it, you can use the same classname but with different content depending on the environment. UserRegistryFactory, therefore, provides an easy way to use a test user registry for authentication in the development and test environments, and then use the actual back-end user registry after the application moves into production.

The UserRegistryFactory, shown in Example A-8, returns the user registry. In this example, an instance of the FileBasedUserRegistry is returned. On the first call to this static method, if the user registry has not yet been created, a new instance is created (and stored) and then returned. Subsequent calls to the static method would return the existing instance.

Using a factory class, such as this, is helpful if the current user registry might change. A different user registry implementation can be created and this factory class changed to instantiate the new user registry and return it. In this case, the calling application is unaffected.

Example A-8 UserRegistryFactory source

```
package com.ibm.itso.saw210.security;

public class UserRegistryFactory {
    private static UserRegistry registry;

    public static synchronized UserRegistry getUserRegistry(){
        if(registry == null){
            registry = new FileBasedUserRegistry();
        }
        return registry;
    }
}
```

Source for FileBasedUserRegistry.java

The FileBasedUserRegistry, shown in Example A-9, is an implementation of the UserRegistry interface class. It is a simple example of a user registry that reads the user name and password from a properties file and authenticates a userid/password combination, passed in as parameters to the authenticate method, against these values.

Example A-9 FileBasedUserRegistry source

```
package com.ibm.itso.saw210.security;

import java.io.IOException;
import java.util.Properties;

public class FileBasedUserRegistry implements UserRegistry {
    private Properties users;
    private String filename = "com/ibm/itso/saw210/security/users.properties";

    public FileBasedUserRegistry() {
        users = new Properties();
        try {
            ClassLoader self = FileBasedUserRegistry.class.getClassLoader();
            users.load(self.getResourceAsStream(filename));
            System.out.println(users);
        }
    }
}
```

```

        catch (IOException io){
            throw new RuntimeException (io);
        }
    }

    public boolean authenticate (String userid, String password){
        if(users.containsKey(userid)){
            String actualPassword = users.getProperty(userid);
            return actualPassword.equals(password);
        }
        return false;
    }
}

```

Source for users.properties

For the FileBasedUserRegistry, the user names and passwords are read from this properties file. The contents, shown in Example A-10, match the user IDs and passwords created in 4.5.1, “Managing access to the Application Center” on page 78 for the Worklight Server, with the addition of the first user ID, which is used in unit testing.

Example A-10 Users defined in the FileBasedUserRegistry

```

worklight=worklight
CSR1=csr1
CSR2=csr2
CSR3=csr3
CSR4=csr4
CSR5=csr5
CSR6=csr6
Tester1=tester1
Tester2=tester2

```

LuggageTracker mobile application files

The following files are those within the mobile application project, CompanyAirlines, consisting of the HTML that is generated from the rich graphical editor and JavaScript files that handle client-side functions. These were created through the steps described in the following sections:

- ▶ 4.8.1, “Creating the user interface” on page 105
- ▶ 4.8.3, “Creating the client-side authentication component” on page 140
- ▶ 4.8.4, “Integrating the solution” on page 142

Source for LuggageTracker.html

The LuggageTracker.html file is the generated HTML file for the user interface. It also is updated to include various JavaScript files that are needed to complete the client-side functionality. The complete HTML file is shown in Example A-11.

Example A-11 LuggageTracker.html file contents

```

<!DOCTYPE HTML>
<html>
<head>
<script>window.$ = window.jQuery = WLJQ;</script>
<title>LuggageTracker</title>

```

```

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script src="jquery.mobile-1.2.0/jquery.mobile-1.2.0.js"></script>
<link href="jquery.mobile-1.2.0/jquery.mobile-1.2.0.css"
    rel="stylesheet">
<link href="jquery.mobile-1.2.0/jquery.mobile.structure-1.2.0.css"
    rel="stylesheet">
<link href="jquery.mobile-1.2.0/jquery.mobile.theme-1.2.0.css"
    rel="stylesheet">
<meta name="viewport"
    content="width=device-width, initial-scale=1.0, maximum-scale=1.0,
    minimum-scale=1.0, user-scalable=0">
<link rel="stylesheet" href="css/LuggageTracker.css">
</head>
<body id="content" style="display: none;">
    <div data-role="page" id="page">
        <div data-role="header" id="header" data-position="fixed">
            <h3>Luggage Tracker</h3>
        </div>
        <div data-role="content" style="padding: 15px">
            <div data-role="content" id="luggageIDInput">
                <label for="text">Luggage ID:</label>
                <input type="text" name="LuggageTrackingID" id="LuggageTrackingID">
                <a href="#" data-role="button" id="scanBarCodeButton" data-theme="e"
onlick="scanBarCode()" data-corners="true">Scan Bar Code</a>
                <a href="#" data-role="button" id="retrieveInfoButton"
data-icon="info" onclick="retrieveInformation()">Retrieve InformationIf</a>
                <a href="#" data-role="button" id="logoutButton"
data-theme="a">Logout</a>
            </div>
            <div data-role="content" id="l_Information">
                <label for="l_Status">Current Status:</label> <input type="text"
name="l_Status" id="l_Status" readonly>
                <label for="l_Owner">Owner Name:</label> <input type="text"
name="l_Owner" id="l_Owner" readonly>
                <label for="l_location">Current Location:</label> <input type="text"
name="l_location" id="l_location" readonly>
                <label for="l_NextLocation">Next Destination:</label> <input
type="text" name="l_NextLocation" id="l_NextLocation" readonly>
                <label for="l_FinalLocation">Final Destination:</label> <input
type="text" name="l_FinalLocation" id="l_FinalLocation" readonly>
                <label for="l_lastUpdate">Last update:</label> <input type="text"
name="l_lastUpdate" id="l_lastUpdate" readonly>
                <label id="l_comments">Comments:</label><textarea id="l_comments"
readonly name="l_comments"></textarea>
                <a href="#" data-role="button" id="changeAddressButton"
onclick="updateDeliveryAddress()">Change Delivery Address</a>
                <a href="#" data-role="button" id="closeButton" data-icon="home"
data-theme="b">Close</a>
            </div>
            <div data-role="content" id="loginForm">
                <label for="usernameInputField">Username:</label><input type="text"
name="usernameInputField" id="usernameInputField">
                <label for="passwordInputField">Password:</label><input
type="password" name="passwordInputField" id="passwordInputField">
                <a href="#" data-role="button" id="loginButton">Login</a>
            </div>
        </div>
    </div>

```



```

        </div>
        <div data-role="content" id="setDeliveryInfo">
            <label for="LuggageTrackingID">Luggage ID:</label><input type="text"
name="LuggageTrackingID" id=LuggageTrackingID">
            <label for="l_DeliveryName">Name:</label><input type="text"
name="l_DeliveryName" id=l_DeliveryName">
            <label for="l_DeliveryAdd1">Address Line 1:</label><input type="text"
name="l_DeliveryAdd1" id=l_DeliveryAdd1">
            <label for="l_DeliveryAdd2">Address Line 2:</label><input type="text"
name="l_DeliveryAdd2" id=l_DeliveryAdd2">
            <label for="l_DeliveryCity">City:</label><input type="text"
name="l_DeliveryCity" id=l_DeliveryCity">
            <label for="l_DeliveryState">State:</label><input type="text"
name="l_DeliveryState" id=l_DeliveryState">
            <label for="l_DeliveryZip">Zip Code:</label><input type="text"
name="l_DeliveryZip" id=l_DeliveryZip">
            <label for="l_DeliveryPhone">Phone Number:</label><input type="text"
name="l_DeliveryPhone" id=l_DeliveryPhone">
            <label id="commentsLabel"
for="l_DeliveryComments">Comments:</label><textarea id="l_DeliveryComments"
name="l_DeliveryComments"></textarea>
            <a href="#" data-role="button" id="saveAndReturnButton">Save and
return</a>
            <a href="#" data-role="button" id="cancelButton">Cancel</a>
        </div>
    </div>
    <div data-role="footer" data-position="fixed" id="footer" data-theme="b">
        <h3></h3>
    </div>
</div>

LuggageTracker
    <!--application UI goes here-->
    <script src="js/initOptions.js"></script>
    <script src="js/LuggageTracker.js"></script>
    <script src="js/messages.js"></script>
    <script src="js/auth.js"></script>
</body>
</html>

```

Source for LuggageTracker.js

The LuggageTracker.js file contains the majority of the functions that are invoked when the buttons on the user interface are clicked. The complete JavaScript file is in Example A-12.

Example A-12 LuggageTracker.js JavaScript file contents

```

/*
 * Licensed Materials - Property of IBM
 * 5725-G92 (C) Copyright IBM Corp. 2006, 2012. All Rights Reserved.
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */
// Worklight comes with the jQuery 1.8.1 framework bundled inside. If you do not
// want to use it, please comment out the line below.
window.$ = window.jQuery = WLJQ;

```

```

function wlCommonInit() {
}

function loadLuggageInformation(result) {
    var lugInfo = result.invocationResult.luggage;
    document.getElementById("notifications").innerHTML = "";
    WL.Client.logActivity("getLuggageInformation adaptor method invoked
sucessfully");
    // Luggage tracking info
    $("#l_Status").val(lugInfo.status);
    $("#l_Owner").val(lugInfo.ownerName);
    $("#l_location").val(lugInfo.currentLocation);
    $("#l_NextLocation").val(lugInfo.nextLocation);
    $("#l_FinalLocation").val(lugInfo.finalDestination);
    $("#l_comments").val(lugInfo.comments);

    // Delivery address Info
    $("#l_DeliveryName").val(lugInfo.deliveryAddress.name);
    $("#l_DeliveryAdd1").val(lugInfo.deliveryAddress.addressLine1);
    $("#l_DeliveryAdd2").val(lugInfo.deliveryAddress.addressLine2);
    $("#l_DeliveryCity").val(lugInfo.deliveryAddress.city);
    $("#l_DeliveryState").val(lugInfo.deliveryAddress.state);
    $("#l_DeliveryZip").val(lugInfo.deliveryAddress.zipCode);
    $("#l_DeliveryPhone").val(lugInfo.deliveryAddress.phoneNumber);
    $("#l_DeliveryComments").val(lugInfo.deliveryAddress.comments);

    $("#luggageIdInput").hide();
    $("#l_Information").show();
}

function luggageNotFound(result) {
    WL.Client.logActivity("error:" + result.invocationResult.errors);
    document.getElementById("notifications").innerHTML = "The call to the server is
unsuccessful";
}

function updateLuggageInformation(luggageId, name, addressLine1, addressLine2,
    city, state, zipCode, phoneNumber, comments) {
    var invocationData = {
        adapter : 'BusinessServicesAdapter',
        procedure : 'addLuggageDeliveryInformation',
        parameters : [ luggageId, name, addressLine1, addressLine2, city,
            state, zipCode, phoneNumber, comments ]
    };
    WL.Client.invokeProcedure(invocationData, {
        onSuccess : reloadLuggageInformation,
        onFailure : luggageNotUpdated,
    });
}

function submitUpdatedDeliveryAddress() {
    document.getElementById("notifications").innerHTML = "";
    updateLuggageInformation($("#LuggageTrackingID").val(),
        $("#l_DeliveryName").val(), $("#l_DeliveryAdd1").val(), $(

```

```

        "#1_DeliveryAdd2").val(), $("#1_DeliveryCity").val(), $(
        "#1_DeliveryState").val(), $("#1_DeliveryZip").val(), $(
        "#1_DeliveryPhone").val(), $("#1_DeliveryComments").val());

    $("#setDeliveryInfo").hide();
    $("#1_Information").show();
}

function luggageNotUpdated(result) {
    var lugInfo = result.invocationResult.luggage;
    WL.Client.logActivity("Luggage delivery address updated");
    $("#1_DeliveryName").val(lugInfo.deliveryAddress.name);
    $("#1_DeliveryAdd1").val(lugInfo.deliveryAddress.addressLine1);
    $("#1_DeliveryAdd2").val(lugInfo.deliveryAddress.addressLine2);
    $("#1_DeliveryCity").val(lugInfo.deliveryAddress.city);
    $("#1_DeliveryState").val(lugInfo.deliveryAddress.state);
    $("#1_DeliveryZip").val(lugInfo.deliveryAddress.zipCode);
    $("#1_DeliveryPhone").val(lugInfo.deliveryAddress.phoneNumber);
    $("#1_DeliveryComments").val(lugInfo.deliveryAddress.comments);
    document.getElementById("notifications").innerHTML = "The call to the server is
    unsuccessful";
}

function reloadLuggageInformation(result) {
    WL.Client.logActivity("failed to update luggage delivery address");
    var lugInfo = result.invocationResult.luggage;
    $("#1_DeliveryName").val(lugInfo.deliveryAddress.name);
    $("#1_DeliveryAdd1").val(lugInfo.deliveryAddress.addressLine1);
    $("#1_DeliveryAdd2").val(lugInfo.deliveryAddress.addressLine2);
    $("#1_DeliveryCity").val(lugInfo.deliveryAddress.city);
    $("#1_DeliveryState").val(lugInfo.deliveryAddress.state);
    $("#1_DeliveryZip").val(lugInfo.deliveryAddress.zipCode);
    $("#1_DeliveryPhone").val(lugInfo.deliveryAddress.phoneNumber);
    $("#1_DeliveryComments").val(lugInfo.deliveryAddress.comments);
    document.getElementById("notifications").innerHTML = "Address updated
    successfully";
}

function retrieveInformation() {
    var luggageID = $("#LuggageTrackingID").val();
    if ((!luggageID) || (luggageID == '')) {
        document.getElementById("notifications").innerHTML = "Tracking Code can't be
        empty";
        return;
    }

    var invocationData = {
        adapter : 'BusinessServicesAdapter',
        procedure : 'getLuggageInformation',
        parameters : [ luggageID ]
    };

    WL.Client.invokeProcedure(invocationData, {
        onSuccess : loadLuggageInformation,
        onFailure : luggageNotFound,
    });
}

```

```

    });
}

function updateDeliveryAddress() {
    var lugID = $("#LuggageTrackingID").val();
    document.getElementById("notifications").innerHTML = "";

    $("#l_Information").hide();
    $("#luggageIdInput").hide();

    $("#l_trackingID").val(lugID);
    $("#setDeliveryInfo").show();
}

function cancelReturn() {
    document.getElementById("notifications").innerHTML = "";

    $("#setDeliveryInfo").hide();
    $("#l_Information").show();
}

function closeApp() {
    $("#l_Status").val("");
    $("#l_Owner").val("");
    $("#l_location").val("");
    $("#l_NextLocation").val("");
    $("#l_FinalLocation").val("");
    $("#l_comments").val("");

    // Delivery address Info
    $("#l_DeliveryName").val("");
    $("#l_DeliveryAdd1").val("");
    $("#l_DeliveryAdd2").val("");
    $("#l_DeliveryCity").val("");
    $("#l_DeliveryState").val("");
    $("#l_DeliveryZip").val("");
    $("#l_DeliveryPhone").val("");
    $("#l_DeliveryComments").val("");

    $("#luggageIdInput").show();
    $("#setDeliveryInfo").hide();
    $("#l_Information").hide();
    document.getElementById("notifications").innerHTML = "";
}

function scanBarCode() {
    if (window.plugins && window.plugins.barcodeScanner) {
        scanBarcode = function() {
            window.plugins.barcodeScanner.scan(function(result) {
                if (!result.cancelled) {
                    $("#LuggageTrackingID").val(result.text);
                    document.getElementById("notifications").innerHTML = "";
                    WL.Client.logActivity("Bar Code Scanner scanned the bar code
successfully ");
                }
            });
        };
    }
}

```

```

    }
    }, function(error) {
        document.getElementById("notifications").innerHTML = error;
        WL.Client.logActivity("Bar Code Scanner Error "+error);
    });
    });
} else {
    scanBarcode = function() {
        document.getElementById("notifications").innerHTML = "No Scanner
available";
        WL.Client.logActivity("No Scanner available");
    };
}

scanBarcode();
}

```

Source for auth.js

The auth.js file (Example A-13) contains the client-side authentication pieces, described in 4.4.2, “Planning client-side authentication” on page 76.

Example A-13 Contents of auth.js file

```

var loginChallengeHandler = WL.Client
    .createChallengeHandler("LuggageTrackerRealm");

var Authenticator = function() {
    $("#loginButton").bind('click', function() {
        var username = $("#usernameInputField").val();
        var password = $("#passwordInputField").val();

        var invocationData = {
            adapter : "LuggageTrackerAdapter",
            procedure : "submitAuthentication",
            parameters : [ username, password ]
        };
        WL.Logger.debug("Password : " + password);
        WL.Logger.debug("Username: " + username);
        loginChallengeHandler.submitAdapterAuthentication(invocationData, {});
    });
    $("#logoutButton").bind('click', onLogout);

    if(WL.Client.isUserAuthenticated("LuggageTrackerRealm"))
    {
        onLogout();
    }
}();

loginChallengeHandler.isCustomResponse = function(response) {
    if (!response || !response.responseJSON || response.responseText === null)
        return false;
    if (typeof (response.responseJSON.authRequired) != 'undefined') {
        return true;
    } else {
        return false;
    }
}

```

```

    }
};

loginChallengeHandler.handleChallenge = function(response) {
    var authRequired = response.responseJSON.authRequired;
    if (authRequired) {
        authFail();
    } else {
        loadApp(response);
        loginChallengeHandler.submitSuccess();
    }
};

function loadApp(response) {
    var isSuccessful = response.responseJSON.isSuccessful;
    WL.Logger.debug("login successful:" + isSuccessful);
    var authRequired = response.responseJSON.authRequired;
    WL.Logger.debug("authRequired value: " + authRequired);
    if (authRequired) {
        $("#luggageIdInput").hide();
        $("#loginForm").show();
    } else {
        $("#luggageIdInput").show();
        $("#loginForm").hide();
    }
}

function authFail() {
    WL.Logger.debug("Login Failed");
    document.getElementById("notifications").innerHTML = "Login Failed";
    WL.Client.reloadApp;
}

function onLogout() {
    var invocationData = {
        adapter : "LuggageTrackerAdapter",
        procedure : "onLogout",
        parameters : []
    };
    loginChallengeHandler.submitAdapterAuthentication(invocationData, {});

    WL.Logger.debug("Logged out");
    $("#luggageIdInput").hide();
    $("#loginForm").show();
}

```

Bar code scanning files

Three separate files are named `barcodescanner.js`:

- ▶ In the common folder of the AirlineShellComponent in the WLAirlineShell project
- ▶ In the android folder of the AirlineShellComponent in the WLAirlineShell project
- ▶ In the iphone folder of the AirlineShellComponent in the WLAirlineShell project

Source for barcodescanner.js in the common folder

The barcodescanner.js file, which is in the common folder, defines the common interface and has an execute method that is overridden by the barcodescanner.js file in each of the native environments. The source is shown in Example A-14.

Example A-14 JavaScript source for the common barcodescanner.js

```
var BarcodeScanner = function() {};  
BarcodeScanner.prototype.scan = function(successCallback, errorCallback) {  
    if (errorCallback == null) {  
        errorCallback = function() {};  
    }  
    if (typeof errorCallback != "function") {  
        console.log("BarcodeScanner.scan failure: failure parameter not a function");  
        return  
    }  
    if (typeof successCallback != "function") {  
        console.log("BarcodeScanner.scan failure: success callback parameter must be a  
function");  
        return  
    }  
    this.execute(successCallback,errorCallback);  
};  
  
BarcodeScanner.prototype.execute = function(successCallback,errorCallback) {  
    errorCallback("Sorry - no Barcodescanner available");  
};  
if(!window.plugins) {  
    window.plugins = {};  
}  
if (!window.plugins.barcodeScanner) {  
    window.plugins.barcodeScanner = new BarcodeScanner();  
}
```

Source for barcodescanner.js in the android folder

The barcodescanner.js file in the android folder is the implementation for the native bar code scanner on the Android platform. The source is shown in Example A-15.

Example A-15 JavaScript implementation for the bar code scanner on Android

```
BarcodeScanner.prototype.execute = function(successCallback,errorCallback) {  
    cordova.exec(successCallback, errorCallback, 'BarcodeScanner', 'scan', []);  
};
```

Source for barcodescanner.js in the iphone folder

The barcodescanner.js file in the iphone folder is the implementation for the native bar code scanner on the iOS platform. The source is shown in Example A-16.

Example A-16 JavaScript implementation for the bar code scanner on iOS

```
BarcodeScanner.prototype.execute = function(successCallback,errorCallback) {  
    cordova.exec(successCallback, errorCallback,  
        'org.apache.cordova.barcodeScanner', 'scan', []);  
};
```

Source for body-top-wlfragment.js in the common folder

The body-top-wlfragment.js file in the common folder is used to inject the script reference for the barcodescanner.js file into the application HTML file. The source is shown in Example A-17.

Example A-17 JavaScript source for body-top-wlfragment.js

```
<!-- This file will be injected at the top part of the <body> element. -->

<!-- Place here any HTML code, links to CSS files, and links to JavaScript files
that should be available for the application. -->

<script src="js/barcodescanner.js"></script>
```

MyLuggage application source files

The MyLuggage mobile application is the second mobile application written by Airline Company A, as described in Chapter 5, “Expanding the solution to consumers” on page 205. The source files provided here are the complete files, which you already created by following the instructions in that chapter. They are provided here as a reference.

Business services adapter files

The following files make up the Business Services adapter, created in 4.7.3, “Creating the business services adapter” on page 99. The BusinessServicesAdapter is part of the AirlineAdapters project in Worklight Studio and is unchanged from the version used with LuggageTracker. It is provided here as a convenience.

Source for BusinessServicesAdapter.xml

The BusinessServicesAdapter.xml file (Example A-18) contains the adapter definition using the <wl:adapter> tag. It defines the type of connection (in this example, HTTP) and also the server name and port to connect to on the back-end. The **server_name** and **server_port** should match your individual environment.

Example A-18 BusinessServicesAdapter.xml contents

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--
    Licensed Materials - Property of IBM
    5725-G92 (C) Copyright IBM Corp. 2011, 2012. All Rights Reserved.
    US Government Users Restricted Rights - Use, duplication or
    disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
-->
<wl:adapter name="BusinessServicesAdapter"
  xmlns:wl="http://www.worklight.com/integration"
  xmlns:http="http://www.worklight.com/integration/http"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <displayName>BusinessServicesAdapter</displayName>
  <description>Provides access to enterprise business services</description>
  <connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>http</protocol>
```



```

        <domain>server_name</domain>
        <port>server_port</port>
    </connectionPolicy>
    <loadConstraints maxConcurrentConnectionsPerNode="2"/>
</connectivity>

    <procedure name="getLuggageInformation" audit="true"/>
    <procedure name="addLuggageDeliveryInformation" audit="true"/>
</wl:adapter>

```

Source for BusinessServicesAdapter-impl.js

The contents of the business services adapter JavaScript implementation file (BusinessServicesAdapter-impl.js) is shown in Example A-19. This file contains the functions that are called when the adapter procedures (defined in BusinessServicesAdapter.xml) are invoked. These functions interact with the back-end services through the invokeHttp() method calls.

Example A-19 JavaScript source contents for BusinessServicesAdapter-impl.js

```

/*
 * Licensed Materials - Property of IBM
 * 5725-G92 (C) Copyright IBM Corp. 2011, 2012. All Rights Reserved.
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */

function getLuggageInformation(luggageId) {
    var input = {
        method : 'get',
        returnerContentType : 'xml',
        path : 'AirlineREST/jaxrs/luggage/'+luggageId,
        transformation : {
            type : 'xslFile',
            xslFile : 'filtered.xsl'
        }
    };

    return WL.Server.invokeHttp(input);
}

function addLuggageDeliveryInformation(lugId, ownerName, addLine1, addLine2,
cityName, stateName, zip, phone, deliveryComments) {
    var input = {
        method : 'post',
        returnerContentType : 'xml',
        senderContentType : 'xml',
        path : 'AirlineREST/jaxrs/luggage/',
        parameters : {lugId : lugId,
            name : ownerName,
            addressLine1 : addLine1,
            addressLine2 : addLine2,
            city : cityName,
            state : stateName,
            zipCode : zip,
            phoneNumber : phone,

```

```

        comments : deliveryComments
    }
};

return WL.Server.invokeHttp(input);
}

```

Source for filtered.xsl

The `filtered.xsl` stylesheet file (Example A-20) is used to filter unnecessary data that is returned from the back-end enterprise business services. Because our back-end services were written specifically for the purposes of this book, the transformation is not necessary. In a real-world scenario, most existing business services return more data than needed for the application so this example was included to show how the data transformation could be done.

Example A-20 Source contents of filtered.xsl file

```

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:h="http://www.w3.org/1999/xhtml">
  <xsl:output method="text"/>
  <xsl:template match="/luggage">
    {
      'luggage': {
        'trackingId' : '<xsl:value-of select="trackingId"/>',
        'ownerName' : '<xsl:value-of select="ownerName"/>',
        'status' : '<xsl:value-of select="status"/>',
        'flightNumber' : '<xsl:value-of select="flightNumber"/>',
        'currentLocation' : '<xsl:value-of select="currentLocation"/>',
        'nextLocation' : '<xsl:value-of select="nextLocation"/>',
        'finalDestination' : '<xsl:value-of select="finalDestination"/>',
        'comments' : '<xsl:value-of select="comments"/>',
        'lastUpdate' : '<xsl:value-of select="lastUpdate"/>',
        'deliveryAddress' : {
          'addressLine1' : '<xsl:value-of
select="deliveryAddress/addressLine1"/>',
          'addressLine2' : '<xsl:value-of
select="deliveryAddress/addressLine2"/>',
          'city' : '<xsl:value-of select="deliveryAddress/city"/>',
          'comments' : '<xsl:value-of select="deliveryAddress/comments"/>',
          'name' : '<xsl:value-of select="deliveryAddress/name"/>',
          'phoneNumber' : '<xsl:value-of
select="deliveryAddress/phoneNumber"/>',
          'state' : '<xsl:value-of select="deliveryAddress/state"/>',
          'zipCode' : '<xsl:value-of select="deliveryAddress/zipCode"/>'
        }
      }
    }
  </xsl:template>
</xsl:stylesheet>

```

Authentication adapter files

The following files are part of the LuggageTracker adapter, which handles authentication, created in 4.7.2, “Creating the authentication adapter” on page 91. LuggageTrackerAdapter is part of the AirlineAdapters project in Worklight Studio and is unchanged from the version used with LuggageTracker. It is provided here as a convenience.

Source for LuggageTrackerAdapter.xml

The LuggageTrackerAdapter.xml file (Example A-21) contains the adapter definition using the <wl:adapter> tag. It defines the type of connection (in this example, HTTP) and also the server name and port to connect to on the back-end. The **server_name** and **server_port** must match your individual environment.

Example A-21 LuggageTrackerAdapter.xml source

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Licensed Materials - Property of IBM
    5725-G92 (C) Copyright IBM Corp. 2011, 2012. All Rights Reserved.
    US Government Users Restricted Rights - Use, duplication or
    disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
-->
<wl:adapter name="LuggageTrackerAdapter"
  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>LuggageTrackerAdapter</displayName>
  <description>LuggageTrackerAdapter</description>
  <connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>http</protocol>
      <domain>server_name</domain>
      <port>server_port</port>
    </connectionPolicy>
    <loadConstraints maxConcurrentConnectionsPerNode="2" />
  </connectivity>

  <procedure name="submitAuthentication" audit="true"/>
  <procedure name="onLogout" audit="true"/>
</wl:adapter>
```

Source for LuggageTrackerAdapter-impl.js

The LuggageTrackerAdapter is the adapter which is used for server-side authentication. It is called by the client-side authentication to validate the credentials provided by the application user. The contents of the implementation JavaScript file (LuggageTrackerAdapter-impl.js) is shown in Example A-22.

Example A-22 LuggageTrackerAdapter-impl.js source

```
/*
 * Licensed Materials - Property of IBM
 * 5725-G92 (C) Copyright IBM Corp. 2006, 2012. All Rights Reserved.
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
```

```

*/

function onAuthRequired(headers, errorMessage){
    WL.Logger.debug("onAuthRequired");
    errorMessage = errorMessage ? errorMessage : null;

    return {
        isLoginSuccessful: false,
        authRequired: true,
        errorMessage: errorMessage
    };
}

function submitAuthentication(username, password){
    WL.Logger.debug("Auth server call submitted");
    WL.Logger.debug("user:" + username);
    WL.Logger.debug("pass:" + password);
    if (username=="worklight" && password === "worklight"){
        WL.Logger.debug("Auth call submitted");
        var userIdentity = {
            userId: username,
            displayName: username,
            attributes: {
                foo: "bar"
            }
        };

        WL.Server.setActiveUser("LuggageTrackerRealm", userIdentity);

        return {
            isLoginSuccessful:true,
            authRequired: false
        };
    }
    WL.Server.setActiveUser("LuggageTrackerRealm", null);
    return onAuthRequired(null, "Invalid login credentials");
}

function onLogout(){
    WL.Server.setActiveUser("LuggageTrackerRealm", null);
    WL.Logger.debug("Logged out");
}

```

Map adapter files

The MyLuggage application obtains hotel information to display on a map using a Cast Iron adapter and a third-party service. It caches the hotel information using WebSphere eXtreme Scale.

Source for MapAdapter.xml

The MapAdapter.xml file contains the adapter definition using the <wl:adapter> tag. It defines the type of connection (in this example, HTTP) and also the server name and port to connect to on the back-end. The contents of this file is shown in Example A-23 on page 355. The **server_name** and **server_port** must match your individual environment.

Example A-23 XML source for MapAdapter adapter

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Licensed Materials - Property of IBM
    5725-G92 (C) Copyright IBM Corp. 2011, 2012. All Rights Reserved.
    US Government Users Restricted Rights - Use, duplication or
    disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
-->
<wl:adapter name="MapAdapter"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wl="http://www.worklight.com/integration"
  xmlns:http="http://www.worklight.com/integration/http"
  xmlns:ci="http://www.worklight.com/integration/ci">

  <displayName>MapAdapter</displayName>
  <description>MapAdapter</description>
  <connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>http</protocol>
      <domain>server_name</domain>
      <port>server_port</port>
    </connectionPolicy>
    <loadConstraints maxConcurrentConnectionsPerNode="2" />
  </connectivity>

  <procedure name="getHotels"/>
  <procedure name="getCachedHotels"/>

</wl:adapter>
```

Source for MapAdapter-impl.js

The contents of the map adapter JavaScript implementation file (MapAdapter-impl.js) is shown in Example A-24. This file contains the functions that are called when the adapter procedures (defined in MapAdapter.xml) are invoked. This includes invoking Cast Iron to retrieve the hotel information and also caching of that information.

Example A-24 JavaScript source for the map adapter implementation

```
function getHotels(latitude, longitude){
  var input = {
    method : 'get',
    appName : '/',
    path : 'map?q=hotel&at='+latitude+','+longitude,
    returnedContentType : 'json'
  };
  return WL.Server.invokeCastIron(input);
}

function getCachedHotels(latitude, longitude) {
  // Create the caching key
  var inputKey = latitude.toFixed(2)+'/'+longitude.toFixed(2);

  // Connect to eXtreme Scale
```

```

    var wLCache = com.ibm.worklight.example.extremescale.WLCache("localhost", 2809,
"Grid", "Map1");

    if (wLCache.getString(inputKey)) { // object already exists, return cached value
        return JSON.parse(wLCache.getString(inputKey));
    }
    else { //Invoke Cast Iron to retrieve hotel information
        var input = {
            method : 'get',
            appName : '/',
            path : 'map?q=hotel&at='+latitude+','+longitude,
            returnedContentType : 'json'
        };

        var result = WL.Server.invokeCastIron(input);

        if(result != null){ // adding new value to the cache
            wLCache.putString(inputKey, JSON.stringify(result));
        }
    }
    return result;
}

```

WebSphere eXtreme Scale caching integration files

To encapsulate the connection to WebSphere eXtreme Scale, data retrieval, and updating the cache, Airline Company A created three Java files:

- ▶ ConnectXS.java
- ▶ ExternalXS.java
- ▶ WLCache.java

Source for ConnectXS.java

The ConnectXS class, shown in Example A-25, is a utility library that acts as a client proxy for connecting to the eXtreme Scale server.

Example A-25 Source code for ConnectXS utility class

```

package com.ibm.worklight.example.extremescale;

import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ConnectException;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.plugins.TransactionCallbackException;

public class ConnectXS {
    private ObjectGrid extremeGrid=null;
    private Session session=null;

```

```

    public ConnectXS(String connectionString, String gridName) throws
ObjectGridRuntimeException
    {
        try {
            ClientClusterContext context =
ObjectGridManagerFactory.getObjectGridManager().connect(connectionString, null,
null);
            this.extremeGrid =
ObjectGridManagerFactory.getObjectGridManager().getObjectGrid(context, gridName);
        } catch (ConnectException e) {
            throw new ObjectGridRuntimeException ("Unable to conect to catalog server
at endpoints:" + connectionString, e);
        }
    }

    public ObjectMap getMap(String mapName) throws ObjectGridException{
        return this.getSession().getMap(mapName);
    }

    public Session getSession() throws TransactionCallbackException,
ObjectGridException{
        if(this.session == null){
            this.session = extremeGrid.getSession();
        }
        return this.session;
    }

    public void closeSession(){
        if(this.session != null){
            this.session.close();
        }
    }
}

```

Source for ExternalXS.java

The ExternalXS class, shown in Example A-26, is a abstraction layer on top of the eXtreme Scale APIs that exposes each of the eXtreme Scale operations as create, read, update and delete (CRUD) methods.

Example A-26 Source code from External XS abstraction layer class

```

package com.ibm.worklight.example.extremescale;

import com.ibm.websphere.objectgrid.DuplicateKeyException;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectMap;

public class ExternalXS {
    private ObjectMap extremeMap;

    public ExternalXS(ObjectMap extremeMap) {
        super();
        this.extremeMap = extremeMap;
    }
}

```

```

    public void put(String key, Object value) throws DuplicateKeyException,
ObjectGridException{
        this.extremeMap.insert(key, value);
    }

    public void post(String key, Object value) throws DuplicateKeyException,
ObjectGridException{
        this.extremeMap.update(key, value);
    }
    public Object get(String key) throws DuplicateKeyException,
ObjectGridException{
        return this.extremeMap.get(key);
    }
    public void delete(String key) throws DuplicateKeyException,
ObjectGridException{
        this.extremeMap.remove(key);
    }
}

```

Source for WLCache.java

The WLCache class, shown in Example A-27, is a further abstraction layer on top of ExternalXS that “understands” the various Worklight Data Objects that are being stored in the cache and acts as a conversion layer for encoding and decoding the data being stored in the cache.

Example A-27 Source code for WLCache abstraction layer class

```

package com.ibm.worklight.example.extremescale;

import java.io.IOException;

import org.mozilla.javascript.Scriptable;
import org.mozilla.javascript.ScriptableObject;

import com.ibm.json.java.JSONArray;
import com.ibm.websphere.objectgrid.ConnectException;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.ws.xs.json.java.JSONObject;
import com.worklight.server.integration.api.JSONObjectConverter;

public class WLCache {
    private ConnectXS connection;
    private ExternalXS api;

    private String connectionServer;
    private String grid;
    private String map;

    public WLCache(String server, String port, String grid, String map){
        this.connectionServer = server + ":" + port;
        this.grid = grid;
        this.map = map;
    }
}

```



```

private void connect() throws ConnectException{
    connection = new ConnectXS(connectionServer, grid);
    ObjectMap objectMap;
    try {
        objectMap = connection.getMap(map);
        api = new ExternalXS(objectMap);
    } catch (ObjectGridException e) {
        throw new ConnectException ("Could not connect to eXtreme Scale Cache",
e);
    }
}

private void disconnect() throws ConnectException{
    if(connection == null && api == null){
        throw new ConnectException("Could not disconnect from a non-connected
eXtreme Scale Cache");
    }
    connection.closeSession();
    connection = null;
    api = null;
}

public void put (String key, Scriptable response) throws ConnectException,
ObjectGridException, IOException{
    this.connect();
    JSONArray jsonArray = (JSONArray)
JSONObjectConverter.scriptableToJSON(response);
    api.put(key, jsonArray.serialize());
    this.disconnect();
}

public void putString (String key, String value) throws ConnectException,
ObjectGridException, IOException{
    this.connect();
    // JSONArray jsonArray = (JSONArray)
JSONObjectConverter.scriptableToJSON(response);
    api.put(key, value);
    this.disconnect();
}

public boolean hasKey(String key) throws ConnectException, ObjectGridException,
IOException{
    this.connect();
    Object result = api.get(key);
    this.disconnect();
    return result != null;
}

public ScriptableObject get (String key) throws ConnectException,
ObjectGridException, IOException{
    try {
        this.connect();
        java.lang.System.out.println("object connected...0");
        String result = (String)api.get(key);
        java.lang.System.out.println("object connected...1");
        JSONArray jsonArray = JSONArray.parse(result);

```

```

        java.lang.System.out.println("object connected...2");
        JSONObject jsonObject = new JSONObject();
        java.lang.System.out.println("object connected...3");
        jsonObject.put("items", jsonArray);
        java.lang.System.out.println("object connected...4");
        ScriptableObject scriptable = (ScriptableObject)
JSONObjectConverter.jsonToScriptable(jsonObject);
        java.lang.System.out.println("retrieving...");
        this.disconnect();
        java.lang.System.out.println("disconnected");
        return scriptable;
    } catch (Exception e) {
        System.out.println(e);
        return null;
    }
}

public String getString (String key) throws ConnectException,
ObjectGridException, IOException{
    try {
        this.connect();
        java.lang.System.out.println("object connected...0");
        String result = (String)api.get(key);
        return result;

    } catch (Exception e) {
        System.out.println(e);
        return null;
    }
}
}

```

Dojo mobile application source files

As described in 5.4.1, “Creating the user interface using Dojo” on page 240, the MyLuggage user interface is written using Dojo Mobile. The following files are in the mobile application project, CompanyAirlines, consisting of the HTML that is generated from the Rich Application Editor and JavaScript files that handle client-side functions.

Source for LuggageTrackerDojo.html

The LuggageTrackerDojo.html file is the generated HTML file for the user interface and includes several JavaScript files that are needed to complete the client-side functionality. The complete HTML file is shown in Example A-28.

Example A-28 Finished HTML for the Dojo-based user interface

```

<!DOCTYPE HTML>
<html>
<head>
<script>window.$ = window.jQuery = WLJQ;</script>
<title>LuggageTrackerDojo</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<meta name="viewport"
    content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">

```

```

<meta name="apple-mobile-web-app-capable" content="yes">
<script type="text/javascript">
    dojoConfig = {
        parseOnLoad: true,
        isDebug: false,
        async: true,
        baseUrl: "",
        packages:[
            { "name": "dojo", "location": "dojo" },
            { "name": "dijit", "location": "dijit" },
            { "name": "dojox", "location": "dojox" },
            { "name": "luggage", "location" : "js" }
        ]
    };
</script>
<script type="text/javascript" src="dojo/dojo.js"></script>
<link rel="stylesheet" href="css/LuggageTrackerDojo.css">
</head>
<body id="content" style="display: none;">
    <h1 data-dojo-type="dojox.mobile.Heading"
        data-dojo-props="label:'My Luggage'" ></h1>
    <div data-dojo-type="dojox.mobile.ScrollableView" id="loginView"
data-dojo-props="selected:true">
        <div data-dojo-type="dojox.mobile.RoundRect">
            <table style="width:100%">
                <tr><td><label for="username">User Name:</label>
                    <input id="username" data-dojo-type="dojox.mobile.TextBox"
style="width:100%"></td></tr>
                <tr><td><label for="password">Password:</label>
                    <input id="password" type="password"
data-dojo-type="dojox.mobile.TextBox" style="width:100%"></td></tr>
                <tr><td><button id="loginBtn" data-dojo-type="dojox.mobile.Button"
style="width:100%">Login</button></td></tr>
            </table>
        </div>
    </div>
    <div data-dojo-type="dojox.mobile.ScrollableView" id="luggageInputView">
        <div data-dojo-type="dojox.mobile.RoundRect">
            <table style="width:100%">
                <tr><td><label for="luggageTrackingId">Luggage ID:</label>
                    <input id="luggageTrackingId" data-dojo-type="dojox.mobile.TextBox"
style="width:100%"></td></tr>
                <tr><td><button id="scanBtn" data-dojo-type="dojox.mobile.Button"
class="mb1RedButton" style="width:100%" >Scan Bar Code</button></td></tr>
                <tr><td><button id="retrieveInformationBtn"
data-dojo-type="dojox.mobile.Button" style="width:100%">Retrieve
Information</button></td></tr>
                <tr><td><button id="logoutBtn" data-dojo-type="dojox.mobile.Button"
style="width:100%">Logout</button></td></tr>
            </table>
        </div>
    </div>
    <div data-dojo-type="dojox.mobile.ScrollableView" id="luggageInformationView">
        <div data-dojo-type="dojox.mobile.RoundRect">
            <table style="width:100%">
                <tr><td><label for="luggageStatus">Current Status:</label>
                    <input id="luggageStatus" data-dojo-type="dojox.mobile.TextBox"
style="width:100%" readonly></td></tr>
                <tr><td><label for="luggageOwner">Owner Name:</label>

```

```

        <input id="luggageOwner" data-dojo-type="dojox.mobile.TextBox"
style="width:100%" readonly></td></tr>
        <tr><td><label for="luggageCurrentLocation">Current Location:</label>
        <input id="luggageCurrentLocation" data-dojo-type="dojox.mobile.TextBox"
style="width:100%" readonly></td></tr>
        <tr><td><label for="luggageNextDestination">Next Destination:</label>
        <input id="luggageNextDestination" data-dojo-type="dojox.mobile.TextBox"
style="width:100%" readonly></td></tr>
        <tr><td><label for="luggageFinalDestination">Final Destination:</label>
        <input id="luggageFinalDestination"
data-dojo-type="dojox.mobile.TextBox" style="width:100%" readonly></td></tr>
        <tr><td><label for="luggageLastUpdate">Last Update:</label>
        <input id="luggageLastUpdate" data-dojo-type="dojox.mobile.TextBox"
style="width:100%" readonly></td></tr>
        <tr><td><label for="luggageComments">Comments:</label>
        <textarea id="luggageComments" data-dojo-type="dojox.mobile.TextArea"
style="width:100%" readonly></td></tr>
        <tr><td><button data-dojo-type="dojox.mobile.Button"
id="changeDeliveryAddressBtn" style="width:100%">Change Delivery Address</button>
        <tr><td><button data-dojo-type="dojox.mobile.Button"
id="closeLuggageInformationViewBtn" class="mb1RedButton"
style="width:100%">Close</button></td></tr>
    </table>
</div>
</div>

<div data-dojo-type="dojox.mobile.ScrollableView" id="luggageDeliveryView">
    <div data-dojo-type="dojox.mobile.RoundRect">
        <table style="width:100%;">
            <tr><td><label for="luggageDeliveryId">Luggage ID:</label>
            <input id="luggageDeliveryId" data-dojo-type="dojox.mobile.TextBox"
style="width:100%;"></td></tr>
            <tr><td><label for="luggageDeliveryOwner">Name:</label>
            <input id="luggageDeliveryOwner" data-dojo-type="dojox.mobile.TextBox"
style="width:100%;"></td></tr>
            <tr><td><label for="luggageDeliveryAddress1">Address Line 1:</label><br/>
            <input id="luggageDeliveryAddress1"
data-dojo-type="dojox.mobile.TextBox" style="width:75%;">
            <button id="hotelMapBtn" data-dojo-type="dojox.mobile.Button"
style="width:20%;">Map</button></td></tr>
            <tr><td><label for="luggageDeliveryAddress2">Address Line 2:</label>
            <input id="luggageDeliveryAddress2"
data-dojo-type="dojox.mobile.TextBox" style="width:100%;"></td></tr>
            <tr><td><label for="luggageDeliveryCity">City:</label>
            <input id="luggageDeliveryCity" data-dojo-type="dojox.mobile.TextBox"
style="width:100%;"></td></tr>
            <tr><td><label for="luggageDeliveryState">State:</label>
            <input id="luggageDeliveryState" data-dojo-type="dojox.mobile.TextBox"
style="width:100%;"></td></tr>
            <tr><td><label for="luggageDeliveryZipCode">Zip Code:</label>
            <input id="luggageDeliveryZipCode" data-dojo-type="dojox.mobile.TextBox"
style="width:100%;"></td></tr>
            <tr><td><label for="luggageDeliveryPhoneNumber">Phone Number:</label>
            <input id="luggageDeliveryPhoneNumber"
data-dojo-type="dojox.mobile.TextBox" style="width:100%;"></td></tr>
            <tr><td><label for="luggageDeliveryComments">Comments:</label>
            <textarea id="luggageDeliveryComments"
data-dojo-type="dojox.mobile.TextArea" style="width:100%;"></td></tr>
            <tr><td><button id="updateDeliveryBtn" data-dojo-type="dojox.mobile.Button"
style="width:100%;">Save and return</button>

```

```

        <tr><td><button id="cancelDeliveryBtn" data-dojo-type="dojox.mobile.Button"
class="mblRedButton" style="width:100%;">Cancel</button></td></tr>
    </table>
</div>
</div>

<h1 data-dojo-type="dojox.mobile.Heading"
    data-dojo-props="fixed: 'bottom'"></h1>

LuggageTrackerDojo
<!--application UI goes here-->
<script src="js/initOptions.js"></script>
<script src="js/LuggageTrackerDojo.js"></script>
<script src="js/messages.js"></script>
</body>
</html>

```

Source for Delivery.js

The Delivery.js JavaScript file handles the client-side functionality for the Change delivery address screen in the Dojo-based MyLuggage application. The complete JavaScript file is shown in Example A-29.

Example A-29 JavaScript source for Delivery.js

```

define(["dojo/on", "dijit/registry"], function(on, registry){

    function getHotelList(position){
        var invocationData = {
            adapter : 'MapAdapter',
            procedure : 'getHotels',
            parameters : [ position.coords.latitude, position.coords.longitude]
        };

        WL.Client.invokeProcedure(invocationData, {
            onSuccess : showHotelInMap,
            onFailure : function(){
                alert("Failed to get the hotel list.");
            }
        });
    };

    function showHotelInMap(response){
        var hotels = {"hotels" : response.invocationResult.results.items};
        var params = {positions: JSON.stringify(hotels)};
        alert(params.positions);
        WL.NativePage.show("com.mapview.HotelMapView", function(data) {
            if (data && data.HotelIndexID) {
                registry.byId("luggageDeliveryAddress1").set("value",
                    hotels.hotels[data.HotelIndexID].vicinity);
            }
        }, params);
    };

    function getUpdatedInformation(){
        var luggageId = registry.byId("luggageDeliveryId").get("value");
        var name = registry.byId("luggageDeliveryOwner").get("value");
        var addressLine1 = registry.byId("luggageDeliveryAddress1").get("value");
        var addressLine2 = registry.byId("luggageDeliveryAddress2").get("value");
        var city = registry.byId("luggageDeliveryCity").get("value");
    }

```

```

        var state = registry.byId("luggageDeliveryState").get("value");
        var zipCode = registry.byId("luggageDeliveryZipCode").get("value");
        var phoneNumber = registry.byId("luggageDeliveryPhoneNumber").get("value");
        var comments = registry.byId("luggageDeliveryComments").get("value");
        return [luggageId, name, addressLine1, addressLine2, city, state, zipCode,
        phoneNumber, comments];
    };

    function updateDeliveryInformation(){
        var params = getUpdatedInformation();
        var invocationData = {
            adapter : 'BusinessServicesAdapter',
            procedure : 'addLuggageDeliveryInformation',
            parameters : params
        };
        WL.Client.invokeProcedure(invocationData, {
            onSuccess : function(){
                alert("Update complete.");
            }
        });

        registry.byId("luggageDeliveryView").performTransition("luggageInformationView", -1,
        "slide");
    },
    onFailure : function(){
        alert("Failed to connect to server.");
    },
    });
};

return {
    initView: function(){
        on(registry.byId("updateDeliveryBtn"), "click", updateDeliveryInformation);
        on(registry.byId("cancelDeliveryBtn"), "click", function(){

registry.byId("luggageDeliveryView").performTransition("luggageInformationView", -1,
"slide");
        });
        on(registry.byId("hotelMapBtn"), "click", function(){
            navigator.geolocation.getCurrentPosition(getHotelList);
        });
    }
};
});

```

Source for Information.js

The Information.js JavaScript file handles the client-side functionality for the Luggage information screen, which switches the visible screen between the Luggage information screen and the Change delivery address screen in the Dojo-based MyLuggage application. The complete JavaScript file is shown in Example A-30.

Example A-30 JavaScript source for Information.js

```

define(["dojo/on", "dijit/registry"], function(on, registry){
    return {
        initView: function(){
            on(registry.byId("changeDeliveryAddressBtn"), "click", function(){

registry.byId("luggageInformationView").performTransition("luggageDeliveryView",
1, "slide");
        }
    }
});

```

```

    });

    on(registry.byId("closeLuggageInformationViewBtn"), "click", function(){
registry.byId("luggageInformationView").performTransition("luggageInputView", -1,
"slide");
    });
    }
    };
});

```

Source for Login.js

The Login.js JavaScript file handles the client-side functionality for the Login screen in the Dojo-based MyLuggage application. The complete JavaScript file is shown in Example A-31.

Example A-31 JavaScript source for Login.js

```

define(["dojo/on", "dijit/registry"], function(on, registry){

    function authDP(username, password){
        $.ajax("http://9.186.9.99:7878/authenticateUser", {
            success: function(){
                var invocationData = {
                    adapter : "LuggageTrackerAdapter",
                    procedure : "submitAuthentication",
                    parameters : [ username, password ]
                };

                loginChallengeHandler.submitAdapterAuthentication(invocationData, {});
                registry.byId("loginView").performTransition("luggageInputView", 1, "slide");
            },
            error: function(xhr, status, error){
                alert("Failed: " + status + " ; " + error);
            },
            headers: {
                "Authorization": "Basic " + Base64.encode(username+":"+password)
            }
        });
    }

    function loginButtonClicked(){
        var username = registry.byId("username").get("value");
        var password = registry.byId("password").get("value");
        if (!username){
            alert("User name cannot be empty.");
            return;
        }
        authDP(username, password);
    }

    return {
        initView: function(){
            on(registry.byId("loginBtn"), "click", loginButtonClicked);
        }
    };
});

```

Source for LuggageTrackerDojo.js

The auto-generated LuggageTrackerDojo.js JavaScript file contains the initialization functions in the Dojo-based MyLuggage application. The complete JavaScript file is shown in Example A-32.

Example A-32 JavaScript source for LuggageTrackerDojo.js

```
/*
 * Licensed Materials - Property of IBM
 * 5725-G92 (C) Copyright IBM Corp. 2006, 2012. All Rights Reserved.
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */

// Worklight comes with the jQuery 1.8.1 framework bundled inside. If you do not
// want to use it, please comment out the line below.
window.$ = window.jQuery = WLJQ;

function wlCommonInit(){
    require([ "dojo/core-web-layer", "dojo/mobile-ui-layer",
              "dojo/mobile-compatible-layer" ], dojoInit);

    /*
     * Application is started in offline mode as defined by a connectOnStartup
     * property in initOptions.js file.
     * In order to begin communicating with Worklight Server you need to either:
     *
     * 1. Change connectOnStartup property in initOptions.js to true.
     *    This will make Worklight framework automatically attempt to connect to
     *    Worklight Server as a part of application start-up.
     *    Keep in mind - this may increase application start-up time.
     *
     * 2. Use WL.Client.connect() API once connectivity to a Worklight Server is
     *    required.
     *    This API needs to be called only once, before any other WL.Client methods
     *    that communicate with the Worklight Server.
     *    Don't forget to specify and implement onSuccess and onFailure callback
     *    functions for WL.Client.connect(), e.g:
     *
     *    WL.Client.connect({
     *    onSuccess: onConnectSuccess,
     *    onFailure: onConnectFailure
     *    });
     */

    // Common initialization code goes here

}

function dojoInit() {
    require([ "dojo", "dijit/registry", "dojo/parser", "dojox/mobile",
              "dojox/mobile/compat",
              "dojox/mobile/deviceTheme", "dojox/mobile/ScrollableView",
              "dojox/mobile/Heading", "dojox/mobile/View",
```



```

        "dojox/mobile/TextBox", "dojox/mobile/Button",
        "dojox/mobile/RoundRect", "dojox/mobile/TextArea" ],
        function(dojo, registry) {
            dojo.ready(function() {
                require(["luggage/Login", "luggage/Search",
                        "luggage/Information", "luggage/Delivery",
                        "luggage/Auth"],
                    function(login, search, information, delivery, auth){
                        login.initView();
                        search.initView();
                        information.initView();
                        delivery.initView();
                        auth.init();
                    });
            });
            window.registry = registry;
        });
    };

```

Source for Search.js

The Search.js JavaScript file handles the client-side functionality to retrieve the luggage information in the Dojo-based MyLuggage application. The complete JavaScript file is shown in Example A-33.

Example A-33 JavaScript source for Search.js

```

define(["dojo/on", "dijit/registry", "dojox/mobile/ProgressIndicator"],
    function(on, registry, indicator){
        function scanBtnClicked(){
            if (window.plugins && window.plugins.barcodeScanner) {
                window.plugins.barcodeScanner.scan(function(result) {
                    if (!result.cancelled) {
                        registry.byId("luggageTrackingId").set("value", result.text);
                    }
                }, function(error) {
                    alert(error);
                });
            } else {
                alert("No Scanner available");
            }
        }

        function retrieveInformationBtnClicked(){
            var id = registry.byId("luggageTrackingId").get("value");
            var prog = indicator.getInstance();
            prog.start();
            WL.Client.invokeProcedure({
                adapter: "BusinessServicesAdapter",
                procedure: "getLuggageInformation",
                parameters: [id]
            }, {
                onSuccess: function(response){
                    prog.stop();
                    var luggage = response.invocationResult.luggage;
                    updateLuggageInformationView(luggage);
                }
            });
        }
    }

```

```

        // Delivery address Info
        var address = luggage.deliveryAddress;
        updateDeliveryInformationView(id, address);

registry.byId("luggageInputView").performTransition("luggageInformationView", 1,
"slide");
    },
    onFailure: function(response){
        prog.stop();
        alert("Failed to connect to the server.");
    }
});
});

function updateLuggageInformationView(luggage){
    registry.byId("luggageStatus").set("value", luggage.status);
    registry.byId("luggageOwner").set("value", luggage.luggageOwner);
    registry.byId("luggageCurrentLocation").set("value", luggage.currentLocation);
    registry.byId("luggageNextDestination").set("value", luggage.nextLocation);
    registry.byId("luggageFinalDestination").set("value",
luggage.finalDestination);
    registry.byId("luggageComments").set("value", luggage.comments);
}

function updateDeliveryInformationView(id, address){
    registry.byId("luggageDeliveryId").set("value", id);
    registry.byId("luggageDeliveryOwner").set("value", address.name);
    registry.byId("luggageDeliveryAddress1").set("value", address.addressLine1);
    registry.byId("luggageDeliveryAddress2").set("value", address.addressLine2);
    registry.byId("luggageDeliveryCity").set("value", address.city);
    registry.byId("luggageDeliveryState").set("value", address.state);
    registry.byId("luggageDeliveryZipCode").set("value", address.zipCode);
    registry.byId("luggageDeliveryPhoneNumber").set("value", address.phoneNumber);
    registry.byId("luggageDeliveryComments").set("value", address.comments);
}

function logoutBtnClicked(){
    registry.byId("luggageInputView").performTransition("loginView", -1, "slide");
};

return {
    initView: function(){
        on(registry.byId("scanBtn"), "click", scanBtnClicked);
        on(registry.byId("retrieveInformationBtn"), "click",
retrieveInformationBtnClicked);
        on(registry.byId("logoutBtn"), "click", logoutBtnClicked);
    }
};
});
});

```

auth.js

The auth.js file, shown in Example A-34, contains the client-side authentication pieces for MyLuggage.

Example A-34 JavaScript for auth.js file

```
var loginChallengeHandler;

define([], function() {
    return {
        init : function() {
            loginChallengeHandler =
WL.Client.createChallengeHandler("LuggageTrackerRealm");

            loginChallengeHandler.isCustomResponse = function(response) {
                if (!response || !response.responseJSON || response.responseText ===
null)
                    return false;
                if (typeof (response.responseJSON.authRequired) != 'undefined') {
                    return true;
                } else {
                    return false;
                }
            };

            loginChallengeHandler.handleChallenge = function(response) {
                var authRequired = response.responseJSON.authRequired;
                if (authRequired){
                    alert("Session is invalid, please log in.");
                }
            };
        }
    };
});
```

application-descriptor.xml

The application-descriptor.xml file for MyLuggage is shown in Example A-35.

Example A-35 XML contents of the application-descriptor.xml for MyLuggage

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Licensed Materials - Property of IBM
      5725-G92 (C) Copyright IBM Corp. 2006, 2012. All Rights Reserved.
      US Government Users Restricted Rights - Use, duplication or
      disclosure restricted by GSA ADP Schedule Contract with IBM Corp. -->
<!-- Attribute "id" must be identical to application folder name -->
<application xmlns="http://www.worklight.com/application-descriptor"
id="LuggageTrackerDojo" platformVersion="5.0.5">
    <displayName>My Luggage</displayName>
    <description>LuggageTrackerDojo</description>
    <author>
        <name>application's author</name>
        <email>application author's e-mail</email>
        <homepage>http://mycompany.com</homepage>
        <copyright>Copyright My Company</copyright>
    </author>
</application>
```

```

</author>
<height>460</height>
<width>320</width>
<mainFile>LuggageTrackerDojo.html</mainFile>
<thumbnailImage>common/images/thumbnail.png</thumbnailImage>
<shell>
    <archive>../../wlshe11/AirlineShellComponent-2.0.wlshe11</archive>
</shell>
<android version="1.0">
    <worklightSettings include="true"/>
    <pushSender key="A1zaSyDsDzT47yweRF1GUI9a7vrIZt4iCVerEkU"
senderId="981490556689"/>
    <security>
        <encryptWebResources enabled="false"/>
        <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png,
jpg, jpeg, gif, mp4, mp3"/>
        <publicSigningKey>Replace this text with the public key of the
certificate with which you sign the APK. For details see the Worklight Developer's
Reference Guide.</publicSigningKey>
    </security>
</android>

<worklightServerRootURL>http://9.111.27.111:9080/luggagetracker</worklightServerRo
otURL>
</application>

```

authenticationConfig.xml

The authenticationConfig.xml file for the application, which defines the authentication realm, is shown in Example A-36.

Example A-36 XML contents for the authenticationConfig.xml file

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<tns:loginConfiguration xmlns:tns="http://www.worklight.com/auth/config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <securityTests>
        <customSecurityTest name="LuggageTrackerCustomSecurityTest">
            <test isInternalUserID="true" realm="LuggageTrackerRealm"/>
        </customSecurityTest>
    </securityTests>

    <realms>
        <realm name="LuggageTrackerRealm" loginModule="LuggageTrackerLoginModule">

            <className>com.worklight.integration.auth.AdapterAuthenticator</className>
            <parameter name="login-function"
value="LuggageTrackerAdapter.onAuthRequired"/>
            <parameter name="logout-function"
value="LuggageTrackerAdapter.onLogout"/>
        </realm>

        <realm loginModule="requireLogin" name="WorklightConsole">
            <className>com.worklight.core.auth.ext.FormBasedAuthenticator</className>
            <onLoginUrl>/console</onLoginUrl>
        </realm>
    </realms>
</loginConfiguration>

```

```

        </realm>

    </realms>

    <loginModules>
        <loginModule name="LuggageTrackerLoginModule">

<className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
        </loginModule>

        <loginModule name="requireLogin">

<className>com.worklight.core.auth.ext.SingleIdentityLoginModule</className>
        </loginModule>

    </loginModules>

</tns:loginConfiguration>

```

Updated LuggageTrackerAdapter

The updated LuggageTrackerAdapter definition XML file is shown in Example A-37 and the JavaScript is shown in Example A-38.

Example A-37 Adapter definition XML for the updated LuggageTrackerAdapter

```

<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter name="LuggageTrackerAdapter"
  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>LuggageTrackerAdapter</displayName>
  <description>LuggageTrackerAdapter</description>
  <connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>http</protocol>
      <domain>localhost</domain>
      <port>8088</port>
    </connectionPolicy>
    <loadConstraints maxConcurrentConnectionsPerNode="2" />
  </connectivity>
  <procedure audit="true" name="submitAuthentication"/>
  <procedure audit="true" name="setUserIdentity"/>
  <procedure audit="true" name="onLogout"/>
</wl:adapter>

```

Example A-38 JavaScript for the updated LuggageTrackerAdapter

```

function onAuthRequired(headers, errorMessage){
  WL.Logger.debug("onAuthRequired");
  errorMessage = errorMessage ? errorMessage : null;

  return {
    isLoginSuccessful: false,
    authRequired: true,

```

```

        errorMessage: errorMessage
    };
}

function submitAuthentication(username, password){
    var reg = com.ibm.itso.saw210.security.UserRegistryFactory.getRegistry();
    var authenticationResult = reg.authenticate(username, password);
    if (authenticationResult){
        var userIdentity = {
            userId: username,
            displayName: username,
            attributes: {
                userType: "CSR"
            }
        };
        WL.Server.setActiveUser("LuggageTrackerRealm", userIdentity);
        return {
            isLoginSuccessful:true,
            authRequired: false
        };
    }
    else {
        WL.Server.setActiveUser("LuggageTrackerRealm", null);
        return onAuthRequired(null, "Invalid login credentials");
    }
}

function onLogout(){
    WL.Server.setActiveUser("LuggageTrackerRealm", null);
    WL.Logger.debug("Logged out");
}

function setUserIdentity(username, password){
    if (username){
        WL.Logger.debug("set UserIdentity call is submitted");
        var userIdentity = {
            userId: username,
            displayName: username,
            attributes: {
                userType: "Passenger"
            }
        };

        WL.Server.setActiveUser("LuggageTrackerRealm", userIdentity);

        return {
            isLoginSuccessful:true,
            authRequired: false
        };
    }
    WL.Server.setActiveUser("LuggageTrackerRealm", null);
    return onAuthRequired(null, "Invalid login credentials");
};

```

Native map files

The native map is used to show hotels on a graphical Google Map on Android devices. It contains several files.

Source for Hotel.java

The Hotel class contains two instance variables, which are used to store the name and position of the hotel. The class also contains the constructor and getter/setter methods. The complete source for the Hotel.java file is shown in Example A-39.

Example A-39 Java source for Hotel.java file

```
package com.mapview;

import com.google.android.gms.maps.model.LatLng;

public class Hotel {
    private String name;
    private LatLng position;

    public Hotel(String name, double latitude, double longitude) {
        this.name = name;
        this.position = new LatLng(latitude, longitude);
    }

    public String getName() {
        return name;
    }

    public LatLng getPosition() {
        return position;
    }
}
```

Source for activity_hotel_map_view.xml

The activity_hotel_map_view.xml file is the layout file that defines the contents of the device screen, similar to how the ScrollView defined the contents of the device screen in the Dojo-based user interface. The layout for the native map will display a Google map with hotels shown. The source of this XML file is in Example A-40.

Example A-40 XML source for the activity_hotel_map_view.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    class="com.google.android.gms.maps.SupportMapFragment"/>
```

Source for HotelMapView.java

The HotelMapView class creates a marker to display on the map for each hotel, and assigns each an index that is used when the user clicks on a hotel to select it from the map. It also handles returning the selected hotel information to the application. The complete source for the HotelMapView.java file is shown in Example A-41.

Example A-41 Java source for HotelMapView.java file

```
package com.mapview;
import java.util.ArrayList;
import java.util.HashMap;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.util.Log;

import com.AirlineShellComponentTest.R;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.GoogleMap.OnInfoWindowClickListener;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

public class HotelMapView extends FragmentActivity implements
    OnInfoWindowClickListener {

    private static final String TAG_HOTELS = "hotels";
    private static final String TAG_HOTEL_NAME = "title";
    private static final String TAG_HOTEL_POSITION = "position";
    private static final String TAG = "HotelMapView";

    private HashMap<String, Integer> markerHotelCorrelation;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // get the json containing the hotel information
        String jsonString = getIntent().getStringExtra("positions");

        // parse the json and store the hotel information in this arrayList
        ArrayList<Hotel> hotels = null;
        try {
            JSONObject json = new JSONObject(jsonString);
            JSONArray jsonHotels = json.getJSONArray(TAG_HOTELS);
            hotels = new ArrayList<Hotel>();

            for (int i = 0; i < jsonHotels.length(); i++) {
                JSONObject jsonHotel = jsonHotels.getJSONObject(i);
```



```

        String name          = jsonHotel.getString(TAG_HOTEL_NAME);
        JSONArray position = jsonHotel.getJSONArray(TAG_HOTEL_POSITION);

        double latitude = position.getDouble(0);
        double longitude = position.getDouble(1);

        Hotel hotel = new Hotel(name, latitude, longitude);
        hotels.add(hotel);
    }
} catch (JSONException e) {
    Log.e(TAG, "Problem dealing with the provided JSON String", e);
}

setContentView(R.layout.activity_hotel_map_view);

if (!hotels.isEmpty()) {
    GoogleMap map =
((SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map)).getMap();

    map.setMapType(GoogleMap.MAP_TYPE_NORMAL);

    // for each hotel create a marker and store the id
    // for simplicity the id is just the index of the hotel in the hotel
list
    markerHotelCorrelation = new HashMap<String, Integer>();
    for (int i = 0; i < hotels.size(); i++) {
        Hotel hotel = hotels.get(i);
        Marker marker = map.addMarker(new
MarkerOptions().position(hotel.getPosition()).title(hotel.getName()));
        markerHotelCorrelation.put(marker.getId(), i);
    }

    map.setOnInfoWindowClickListener(this);

map.animateCamera(CameraUpdateFactory.newLatLngZoom(hotels.get(0).getPosition(),
10));
    }
}

@Override
public void onInfoWindowClick(Marker marker) {
    // get the index of the selected hotel
    int indexID = markerHotelCorrelation.get(marker.getId());

    // create Intent to add the index of the selected hotel to
    Intent resultIntent = new Intent();
    resultIntent.putExtra("HotelIndexID", Integer.valueOf(indexID));

    // set the result and return back to the caller
    setResult(RESULT_OK, resultIntent);
    finish();
}
}

```

Source for AndroidManifest.xml

Every Android application must have an `AndroidManifest.xml` file in its root directory, which contains information about the application to the Android system. The complete `AndroidManifest.xml` for `MyLuggage` is in Example A-42.

Example A-42 Source for the `AndroidManifest.xml` file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.AirlineShellComponent"
    android:versionCode="1"
    android:versionName="1.0">

    <supports-screens
        android:smallScreens="false"
        android:normalScreens="true"
        android:largeScreens="false"
        android:resizeable="false"
        android:anyDensity="false"
    />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>

    <!-- Push permissions -->
    <permission android:name="com.AirlineShellComponent.permission.C2D_MESSAGE"
        android:protectionLevel="signature" />
    <uses-permission
        android:name="com.AirlineShellComponent.permission.C2D_MESSAGE" />
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.USE_CREDENTIALS" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <!-- Barcode Scanner permissions -->
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.FLASHLIGHT" />

    <!-- Google Maps permissions -->
    <permission android:name="com.AirlineShellComponent.permission.MAPS_RECEIVE"
        android:protectionLevel="signature" />

    <uses-permission android:name="com.AirlineShellComponent.permission.MAPS_RECEIVE"
    />
    <uses-permission
        android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <uses-feature android:glEsVersion="0x00020000" android:required="true" />

    <application
        android:label="@string/app_name"
        android:debuggable="true"
```

```

        android:icon="@drawable/icon" >
        <activity android:name=".AirlineShellComponent"
            android:label="@string/app_name"
            android:configChanges="orientation|keyboardHidden"
            android:launchMode="singleTask">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action
                    android:name="com.AirlineShellComponent.AirlineShellComponent.NOTIFICATION" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>

        <!-- Barcode Scanner related Activities -->
        <activity
            android:name="com.google.zxing.client.android.CaptureActivity"
            android:configChanges="orientation|keyboardHidden"
            android:exported="false"
            android:screenOrientation="landscape"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
            android:windowSoftInputMode="stateAlwaysHidden" >
            <intent-filter>
                <action android:name="com.phonegap.plugins.barcodescanner.SCAN" />

                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.google.zxing.client.android.encode.EncodeActivity"
            android:label="@string/share_name" >
            <intent-filter>
                <action android:name="com.phonegap.plugins.barcodescanner.ENCODE"
/>

                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>

        <!-- Preference Activity -->
        <activity
            android:name="com.worklight.common.WLPreferences"
            android:label="Worklight Settings">
        </activity>
        <!-- Push service -->
        <!-- In order to use the c2dm library, an application must declare a class
with the name C2DMReceiver, in its own package, extending
com.google.android.c2dm.C2DMBaseReceiver
It must also include this section in the manifest, replacing
"com.google.android.apps.chrometophone" with its package name. -->
        <service android:name=".GCMIntentService" />

```

```

        <!-- Only google service can send data messages for the app. If permission
is not set - any other app can generate it -->
        <receiver android:name="com.google.android.gcm.GCMBroadcastReceiver"
android:permission="com.google.android.c2dm.permission.SEND">
            <!-- Receive the actual message -->
            <intent-filter>
                <action android:name="com.google.android.c2dm.intent.RECEIVE" />
                <category android:name="com.AirlineShellComponent" />
            </intent-filter>
            <!-- Receive the registration id -->
            <intent-filter>
                <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
                <category android:name="com.AirlineShellComponent" />
            </intent-filter>
        </receiver>
        <activity android:name="com.mapview.HotelMapView"></activity>
        <meta-data android:name="com.google.android.maps.v2.API_KEY"
            android:value="<YOUR_GOOGLE_MAPS_API_KEY>" />

    </application>
    <uses-sdk android:minSdkVersion="17" />
</manifest>

```



Additional material

This book 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. Point your web browser to the following location:

<ftp://www.redbooks.ibm.com/redbooks/SG248117>

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, SG248117.

Using the web material

The additional web material that accompanies this book includes the following files:

<i>File name</i>	<i>Description</i>
NokiaHere.par	Compressed exported IBM WebSphere Cast Iron project
NokiaHere.zip	Compressed IBM WebSphere Cast Iron project source code

Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and download the contents of the web material files into this folder. Extract the contents of the files into this folder to access the IBM WebSphere Cast Iron orchestration.

Related publications

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

IBM Redbooks

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.

- ▶ *Elastic Dynamic Caching with the IBM WebSphere DataPower XC10 Appliance*, REDP-4851
- ▶ *Enabling Mobile Apps with IBM Worklight Application Center*, REDP-5005
- ▶ *Enterprise Caching in a Mobile Environment*, TIPS0953
- ▶ *Enterprise Caching Solutions using WebSphere DataPower SOA Appliances and IBM WebSphere eXtreme Scale*, SG24-8043
- ▶ *Getting Started with IBM WebSphere Cast Iron Cloud Integration*, SG24-8004
- ▶ *IBM WebSphere Cast Iron Introduction and Technical Overview*, REDP-4840
- ▶ *Scalable, Integrated Solutions for Elastic Caching Using IBM WebSphere eXtreme Scale*, SG24-7926
- ▶ *User's Guide to WebSphere eXtreme Scale*, SG24-7683

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

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- ▶ IBM Worklight Version 5.0.5 Information Center
<http://pic.dhe.ibm.com/infocenter/wrklight/v5r0m5/index.jsp>
- ▶ IBM Worklight getting started
<https://www.ibm.com/developerworks/mobile/worklight/getting-started/index.html>
- ▶ Eight steps to IBM Worklight mobile application development
http://www.ibm.com/developerworks/websphere/techjournal/1210_chen/1210_chen.html
- ▶ IBM Worklight
<http://www.ibm.com/software/mobile-solutions/worklight/>
- ▶ IBM white paper: *Native, web or hybrid mobile-app development*
<ftp://public.dhe.ibm.com/software/pdf/mobile-enterprise/WSW14182USEN.pdf>

- ▶ IBM white paper: *Mobility is moving fast. To stay in control, you have to prepare for change*
<http://public.dhe.ibm.com/common/ssi/ecm/en/tiw14132usen/TIW14132USEN.PDF>
- ▶ IBM white paper: *A mobile application development primer*
<http://public.dhe.ibm.com/common/ssi/ecm/en/raw14302usen/RAW14302USEN.PDF>
- ▶ IBM data sheet: IBM Worklight
<http://public.dhe.ibm.com/common/ssi/ecm/en/wsd14109usen/WSD14109USEN.PDF>
- ▶ IBM white paper: *IBM Worklight Technology overview*
<http://public.dhe.ibm.com/common/ssi/ecm/en/wsw14181usen/WSW14181USEN.PDF>
- ▶ IBM white paper: *Improve your mobile application security with IBM Worklight*
<http://public.dhe.ibm.com/common/ssi/ecm/en/wsw14198usen/WSW14198USEN.PDF>
- ▶ User interface design for the mobile web
<http://www.ibm.com/developerworks/web/library/wa-interface/index.html>
- ▶ WebSphere Cast Iron Cloud integration
<http://www.ibm.com/software/integration/cast-iron-cloud-integration/features>
- ▶ IBM Endpoint Manager for Mobile Devices
<http://www-142.ibm.com/software/products/us/en/ibmendpmanaformobidevi/>
- ▶ WebSphere DataPower SOA Appliances
<http://www.ibm.com/software/integration/datapower>
- ▶ WebSphere DataPower Service Gateway XG45
<http://www.ibm.com/software/integration/datapower/XG45/>
- ▶ WebSphere DataPower XC10 Appliance
<http://www.ibm.com/software/webservers/appserv/xc10/>
- ▶ WebSphere eXtreme Scale
<http://www.ibm.com/software/webservers/appserv/extremescale/>
- ▶ jQuery Mobile
<http://jquerymobile.com>
- ▶ IBM Installation Manager Version 1 Release 5 Information Center
<http://pic.dhe.ibm.com/infocenter/install/v1r5/index.jsp>
- ▶ IBM Worklight system requirements
<http://www.ibm.com/support/docview.wss?uid=swg27024838>
- ▶ IBM Worklight Server V5.0.5 Information Center - Installing Worklight Server
http://pic.dhe.ibm.com/infocenter/wrklight/v5r0m5/index.jsp?topic=%2Fcom.ibm.worklight.help.doc%2Fstart%2Fr_install_wl_server.html
- ▶ Eclipse information about Retina Display with IBM Worklight Studio
https://bugs.eclipse.org/bugs/show_bug.cgi?id=382972#c4
- ▶ Get the Android SDK
<http://developer.android.com/sdk/index.html>

- ▶ IBM Information Center - WebSphere Cast Iron Hypervisor Edition
http://publib.boulder.ibm.com/infocenter/wci/v6r1m0/topic/com.ibm.websphere.cast_iron.VAuserguide.doc/VA_about_VA.html
- ▶ IBM WebSphere eXtreme Scale V8.5 Information Center
http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Fwelcome%2Fwelcome_xs.html
- ▶ Responsive Web Design: What It Is and How To Use It
<http://coding.smashingmagazine.com/2011/01/12/guidelines-for-responsive-web-design/>
- ▶ Responsive Web Design
<http://www.alistapart.com/articles/responsive-web-design/>
- ▶ Nokia HERE and Nokia Places API
▶ <http://developer.here.com/docs/places/>
- ▶ Nokia HERE registration
<http://developer.here.com/docs/places/#common/credentials.html>
- ▶ Google Play services
<http://developer.android.com/google/play-services/setup.html>
- ▶ Google Maps documentation for Android devices
<https://developers.google.com/maps/documentation/android>
- ▶ Google Maps API documentation and registration
https://developers.google.com/maps/documentation/android/start#getting_the_google_maps_android_api_v2

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



Extending Your Business to Mobile Devices with IBM Worklight

(0.5" spine)
0.475" <-> 0.873"
250 <-> 459 pages



Extending Your Business to Mobile Devices with IBM Worklight



See how to build, run, manage, and integrate mobile applications with IBM Worklight

Explore how to quickly integrate mobile applications with cloud services

Learn to use IBM Worklight for developing mobile applications

The mobile industry is evolving rapidly. An increasing number of mobile devices, such as smartphones and tablets, are sold every year and more people are accessing services from a mobile device than ever before. For an enterprise, this can mean that a growing number of customers, business partners, and even employees now expect to access services on a mobile channel. This opens new opportunities for the business but also presents new challenges, both in terms of business processes and information technology (IT) infrastructure.

IBM Worklight is an open mobile application platform. It helps organizations of all sizes to efficiently develop, connect, run, and manage HTML5, hybrid, and native applications. IBM Worklight provides the essential elements needed for complete mobile application development, deployment, and management within a business.

This IBM Redbooks publication provides information necessary to design, develop, deploy, and maintain mobile applications using IBM Worklight Version 5.0.5. It includes information about decision points that the IT organization will need to make, the roles that are involved in a mobile strategy and the responsibilities of the individuals in those roles. It also describes integration points with other IBM products that can enhance the mobile solution. This book has two parts:

- ▶ Part 1 is for a business-oriented IT audience and addresses business aspects of the mobile industry. It is for the IT architect or CTO, who can translate business needs into information technology solutions
- ▶ Part 2 is intended for a technical audience, including application developers, testers, and system administrators.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

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

For more information:
ibm.com/redbooks

SG24-8117-00

ISBN 0738438448